UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

ESCOLA TÉCNICA SUPERIOR DE ENXEÑARÍA

Herramienta para evaluar la conformidad de procesos de negocio

Autor:
Jacobo Casas Ramos
Directores:
Manuel Lama Penín
Manuel Mucientes Molina

Grao en Enxeñaría Informática

Julio 2019

Traballo de Fin de Grao presentado na Escola Técnica Superior de Enxeñaría da Universidade de Santiago de Compostela para a obtención do Grao en Enxeñaría Informática

Dr. Manuel Lama Penín, Profesor do Departamento de Electrónica e Computación da Universidade de Santiago de Compostela, e **Dr. Manuel Mucientes Molina**, Profesor do Departamento de Electrónica e Computación da Universidade de Santiago de Compostela,

INFORMAN:

Que a presente memoria, titulada *Herramienta para evaluar la conformidad de procesos de negocio*, presentada por **D. Jacobo Casas Ramos** para superar os créditos correspondentes ao Traballo de Fin de Grao da titulación de Grao en Enxeñaría Informática, realizouse baixo nosa dirección no Departamento de Electrónica e Computación da Universidade de Santiago de Compostela.

E para que así conste aos efectos oportunos, expiden o presente informe en Santiago de Compostela, a 28 de junio de 2019:

O director,	O codirector,	O alumno,
Manuel Lama Penín	Manuel Mucientes Molina	Jacobo Casas Ramos

Agradecimentos

A mis tutores, por proporcionarme la posibilidad de desarrollar este proyecto con ellos, y al resto de profesores que me permitieron conseguir los conocimientos necesarios para llegar a este momento.

A la familia y a los amigos, que me apoyaron en todo momento no dejando que me rindiese.

Resumen

Las técnicas de comprobación de conformidad de procesos (*conformance checking*) verifican si las trazas de ejecución de un procesos son consistentes con el modelo que describe su comportamiento [6]. Sin embargo, en ocasiones los usuarios están interesados en consultar si un submodelo se verifica y/o si ciertas condiciones temporales o restricciones sobre indicadores clave de negocio han tenido lugar.

Para resolver este problema se han propuesto trabajos basados en una aproximación declarativa[1] y/o en chequeo de modelos[2]. El principal problema de estas aproximaciones es su falta de flexibilidad a la hora de definir submodelos de procesos que contienen cualquier estructura de control y añadir a dichos modelos condiciones temporales complejas sobre actividades y condiciones sobre indicadores clave de negocio. En esta memoria se abordará este problema con una aproximación basada en técnicas de conformidad de procesos y en tecnologías de análisis masivo de datos (*Big Data*).

[1] *Discovery of Multi-perspective Declarative Process Models*, pp. 87–103 [7]
[2] *Analysis of Users' Behavior in Structured e-Commerce Websites*, pp. 11941–11958 [12]

Índice general

Capítulo 1

Introdución

1.1. Motivación

Hoy en día existen una gran cantidad de registros de todo tipo que guardan eventos de los cuales se podría extraer mucha información útil con las herramientas adecuadas, mejorando los procesos actuales. En esto consiste la minería de procesos y son las partes en las que se divide la misma: descubrir, monitorizar y mejorar procesos reales[1].

En minería de procesos, se parte de un registro de eventos y se aplica un algoritmo de descubrimiento de procesos, obteniendo modelos de los procesos. Después se realiza la comprobación de procesos (conformance checking), normalmente sobre los procesos descubiertos en la etapa anterior y sobre los nuevos registros que se obtienen, para los que se calcula el número de trazas del registro que realmente cumplen el modelo que se presenta. Este proceso lleva directamente al último de la minería de procesos, que es la mejora de los mismos. Los resultados obtenidos con la herramienta desarrollada sirven para detectar fallos en los procesos descubiertos o resultados no esperados en procesos definidos por los usuarios, para después buscar mejoras que aplicar al proceso de negocio.

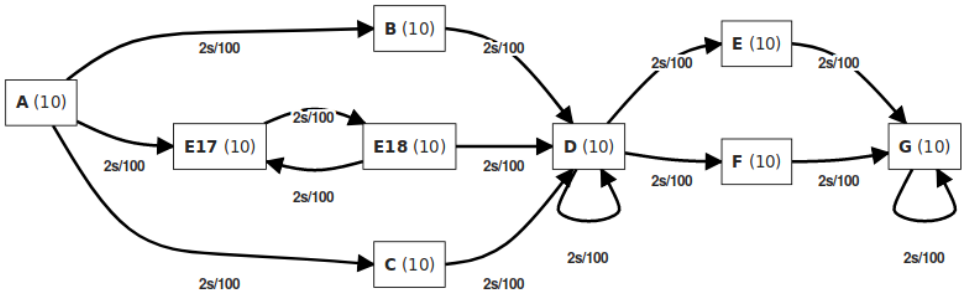


Figura 1.1: Ejemplo de consulta

El uso más básico del algoritmo que se desarrolla en este proyecto consiste en

[1] *Process Mining Manifesto* p. 1 [1]

proporcionarle un registro y un modelo. Un ejemplo de la representación gráfica de un modelo se puede ver en la Figura 1.1. Entonces intentará ejecutar las trazas que se encuentran en el registro sobre el modelo, obteniendo las que lo cumplen. Si se configura para ello, el algoritmo también permite ver los puntos en los que fue necesario realizar una adaptación para conseguir que el modelo acepte la traza (como saltar una actividad de la traza o en el modelo si se permite). Las diferencias de esta herramienta con otras ya existentes en el campo son:

Big Data. Siempre se tiene en cuenta que los registros que se usan pueden llegar a ser muy pesados, por ejemplo evitando importar todo el registro en memoria antes de guardarlo en la base de datos, sino que cada traza leída se inserta directamente a la base de datos. Además, se adapta el algoritmo para poder ser ejecutado en un clúster, con paralelización a nivel de traza. En un registro suele existir una gran cantidad de trazas por lo que la paralelización es muy buena, necesitando una sincronización para repartir las trazas (la cual es mínima cuando estas se encuentran en una instalación de *MongoDB* distribuida entre los nodos del clúster), y otra sincronización para agregar los resultados, cuyo peso depende de las extensiones al algoritmo que se apliquen, pero por defecto sería muy poca por tratarse de una suma de los totales de trazas que han superado la comprobación con la consulta. Para permitir este procesado en un clúster se usa *Spark*.

Modos de ejecución. El algoritmo se puede aplicar en modo *estricto*, que necesita que la consulta se cumpla tal cual en la traza; un modo *relajado* que permite ignorar eventos que sobran en la traza ya que no interesan para la consulta (esto se consigue permitiendo los movimientos solo en traza al utilizar *token replay*); y un modo aún más relajado, que añade un parámetro de *fitness* mínimo que debe superarse en cada paso del algoritmo para poder continuar, y este *fitness* se obtiene a partir de los costes definibles por el usuario (se explicará más profundamente en el primer incremento). Además tiene la ventaja de que se aplica sobre consultas que el usuario puede definir. Esto significa que además de trabajar con modelos completos descubiertos previamente, el algoritmo puede ejecutarse sobre un submodelo que le interese al usuario. Entonces, esta herramienta proporciona mucha flexibilidad no presente en otras.

Restricciones con tiempo. Se permite acceder a los tiempos de inicio y fin de los eventos, además de sus ciclos de vida (iniciar, asignar, suspender, resumir...) en un lenguaje de restricciones que se aplica a cada movimiento de marcas en el algoritmo. El lenguaje de restricciones devuelve un valor lógico verdadero o falso indicando si el algoritmo puede continuar estudiando la traza por esta parte del modelo o debe buscar otra opción, pero también puede devolver un entero que indica el coste del movimiento de la marca. Se explicará la utilidad de este coste en el primer incremento.

Restricciones con *KPIs*. Los *KPIs* o indicadores clave de negocio son los factores o variables que se considera que afectan al negocio. Tienen valores asociados en distintos puntos del registro y no tienen por qué asociar

siempre valores a todos los eventos del mismo. Esta herramienta permite definir restricciones sobre estos valores, ya estén asociados al evento procesado, a la traza procesada o sean valores globales del registro, aplicables mediante el mismo lenguaje que el que define las restricciones de tiempo.

Además, la herramienta desarrollada será una aplicación completa, lo que permite ofrecer servicios para realizar todas estas tareas de forma remota y que usuarios puedan acceder a todas estas funciones desde una interfaz gráfica fácil de usar y aprovechando todas las características desarrolladas para el algoritmo, incluyendo capacidad para importar y exportar registros y consultas en distintos formatos, crear consultas y restricciones gráficamente desde la interfaz...

Un ejemplo de caso de uso podría ser en una tienda, en la que interesa saber en qué casos de los que se han guardado en sus registros, un usuario compró determinado producto (guardado en una variable asociada al evento de compra) antes de la compra del mismo producto por el mismo usuario, y con un límite de tiempo de 7 días. Para conseguir estos datos, se ejecuta el algoritmo con:

Registro. Se supone que se dispone de un registro CSV con datos como estos:
`Usuario,Actividad,Final,Producto,Cantidad,Precio`
`Usuario01,Compra,2019-05-27T17:04:28,Naranjas,10,1.5`
`Usuario01,Devuelve,2019-05-27T17:15:56,Naranjas,3,`
`Usuario01,Compra,2019-05-29T09:04:34,Naranjas,10,1.5`
Como se puede ver no existen datos de KPIs para todos los eventos. Se usa como identificador de la traza el identificador del usuario, de forma que en este registro sólo existiría una traza formada por los 3 eventos.

Consulta. Compra → Compra, donde Compra es el nombre de la actividad. Esto se puede definir mediante el lenguaje que se desarrollará para la construcción sencilla de consultas como: `SEQ(Compra, Compra)`. Esto indica que se deben realizar dos tareas de Compra, pero no define las restricciones necesarias, aunque al usar el identificador de usuario como identificador de traza ya se asegura que se trata de dos compras del mismo usuario.

Modo. Relajado, ya que el usuario puede realizar cualquier otra tarea entre las compras y queremos seguir obteniendo resultados en estos casos.

Restricciones. Existen dos restricciones, entre las cuales se debe aplicar un AND. Para diferenciar los eventos en el lenguaje de restricciones, se realiza una transformación automática de los eventos de la traza cuando estos se repiten: el más reciente mantiene el nombre mientras que a los anteriores se añaden números al final. Por lo tanto el lenguaje de restrcciones podría ser $Compra[\text{“}producto''] = Compra2[\text{“}producto''] \ AND \ (Compra[\text{“}end''] = NULL \ OR \ Compra2[\text{“}end''] = NULL \ OR \ Compra[\text{“}end''] < Compra2[\text{“}end''] + 7 * 86400000)$, donde *producto* es un *KPI* asociado al evento de Compra (podría contener una cadena de texto o un entero indicando el id, por ejemplo) y *end* es la forma de acceder al tiempo de finalización de la tarea, el cual se almacena en milisegundos al importar el

registro (aunque el formato de la fecha guardado en el registro sea otro). Para la segunda condición es necesario comprobar que los datos no sean *null* porque en los pasos iniciales del algoritmo aún no se han encontrado los dos eventos y estos accesos devuelven *null*, pero es necesario seguir buscando hasta encontrar el segundo evento en estos casos. Esto también se puede representar en dos o más líneas ya que el operador lógico entre múltiples líneas es configurable.

Más configuración. Se configurará el algoritmo para registrar las trazas que cumplen la consulta, ya que nos interesa saber qué usuarios son los que cumplen esta consulta. Por defecto esta extensión no está activada porque implica una sincronización final en la que se agregan las listas de todos los nodos, lo que resulta más pesado que simplemente sumar los números de trazas que fueron aceptadas por la consulta.

1.2. Objetivos

El objetivo principal es el desarrollo de una herramienta software para la evaluación de la conformidad sobre modelos de negocio a través de consultas realizadas por los usuarios. Estas consultas son, a su vez, submodelos sobre los que el usuario desea conocer o verificar su cumplimiento en el registro de eventos, obteniendo como resultado el conjunto de trazas que cumplen dichos submodelos. Para alcanzar este objetivo general se definen los siguientes subobjetivos:

- Desarrollo de un algoritmo basado en técnicas de minería de procesos que permita evaluar la conformidad de un conjunto de submodelos definidos por el usuario, donde la conformidad está relacionada con el grado de cumplimiento de la estructura del submodelo sobre el conjunto de trazas de entrada.
- Extender el algoritmo de conformidad para incluir la especificación de relaciones temporales entre las actividades que componen el submodelo de entrada.
- Extender el algoritmo de conformidad para incluir la especificación de restricciones sobre los indicadores clave de negocio entre distintas actividades de la consulta.
- Desarrollo de una interfaz gráfica web avanzada a través de la cual, por una parte, se puedan definir los submodelos, las restricciones temporales entre actividades y las condiciones sobre indicadores clave de negocio, y, por otra parte, se puedan visualizar los resultados. Para ello también se debe desarrollar un servidor web que ofrezca una API de servicios REST para que los usuarios puedan utilizar este algoritmo de forma remota.

1.3. Estructura del documento

Al tratarse de un proyecto desarrollado por incrementos, hay ciertos capítu-

los que es necesario repetir, ya que las etapas los capítulos siguientes se separan por incremento al que pertenecen y, para cada uno de ellos, se desarrolla la planificación, presupuestos, especificación de requisitos, diseño, etc.

Capítulo 1: Introdución: el presente capítulo, es una breve introducción al trabajo realizado.

Capítulo 2: Análisis de requisitos: requisitos identificados en la etapa de análisis preliminar del proyecto.

Capítulo 3: Gestión del proyecto: describe la gestión realizada a lo largo del proyecto, donde se incluye la gestión del alcance, el análisis de riesgos, metodología, planificación de tiempo y cálculo de presupuestos entre otros.

Capítulo 4: Arquitectura: diseño preliminar del sistema que incluye la interacción entre los incrementos. Este capítulo es un resumen, que describe la forma en la que interaccionan los incrementos, mientras que en cada implemento se expandirá la descripción de la arquitectura implementada.

Capítulos 5, 6, 7 y 8: Incrementos: describen el trabajo realizado en cada incremento.

- Capítulos 5.1, 6.1 y 7.1: Arquitectura: definen la arquitectura que se implementa en el incremento, apoyandose en distintos diagramas.
- Capítulos 5.2, 6.2, 7.2 y 8.1: Diseño e implementación: explican el diseño y la implementación para realizar las funcionalidades propuestas.
- Capítulos 5.3, 6.3 y 8.2: Pruebas: se detallan las pruebas realizadas, los casos de prueba y el estudio de los resultados de estas, con informes ejecutivos finales.

Capítulo 9: Conclusiones: las conclusiones del proyecto completo. Incluyen los puntos de mejora, ampliaciones y otro trabajo futuro que se podría hacer relacionado con el proyecto.

Apéndice A: Manuales técnicos: describen en detalle la forma de uso de la aplicación como biblioteca para otras aplicaciones o desplegar la aplicación, enfocada a desarrolladores, y la instalación del servidor.

Apéndice B: Manual de usuario: describe en detalle la forma de uso de la aplicación, enfocada a usuarios.

Apéndice C: Incremento 1: Casos de prueba: por su elevada cantidad se incluyen en un anexo.

Apéndice D: Integración continua: configuración realizada para el sistema de integración continua, que muestra la instalación, prueba, generación de informes de resultados y cobertura, despliegue en el repositorio nexus y muchas otras tareas realizadas automáticamente en la Integración Continua (CI) de forma replicable gracias al uso de Docker.

Apéndice E: Glosario: describe los términos técnicos utilizados a lo largo de esta memoria, que disponen de referencias al propio glosario.

Apéndice F: Bibliografía: las fuentes usadas para desarrollar este proyecto.

Capítulo 2

Análisis de requisitos

El PMBOK define la recopilación de requisitos como:

> "Recopilar Requisitos es el proceso de determinar, documentar y gestionar las necesidades y los requisitos de los interesados para cumplir con los objetivos del proyecto. El beneficio clave de este proceso es que proporciona la base para definir y gestionar el alcance del proyecto, incluyendo el alcance del producto.
>
> El éxito del proyecto depende directamente de la participación activa de los interesados en el descubrimiento y la descomposición de las necesidades en requisitos, y del cuidado que se tenga al determinar, documentar y gestionar los requisitos del producto, servicio o resultado del proyecto."[17]

Este es un proceso clave en el proyecto que podrá determinar si es exitoso o no, por lo que es importante que los interesados participen en él activamente. Se realizaron reuniones con los interesados para definir claramente estos requisitos. La descripción y especificación se realizará mediante tablas basadas en las proporcionadas por las herramienta REM[1].

2.1. Diagrama de contexto

Se comienza el capítulo con la representación del diagrama de contexto. El sistema se irá especificando cada vez más en las secciones siguientes. Este muestra a alto nivel como interactúa el sistema, representado en el centro, con las entidades relacionadas, mostrando el flujo de datos. Este se puede ver en la Figura 2.1.

[1]Herramienta utilizada en Ingeniería del Software para descripción de requisitos [22].

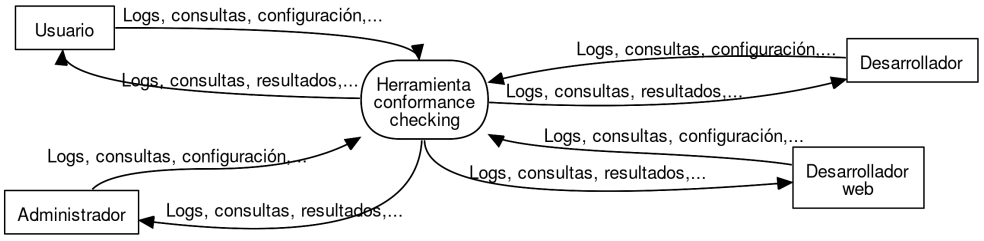


Figura 2.1: Diagrama de contexto.

2.2. Casos de uso

Los módulos del sistema se describen mediante diagramas de casos de uso y sus especificaciones claras. Estos abarcan todo el sistema desde el punto de vista del usuario, por lo que hay una carga importante de la interfaz y acciones que pueden realizar usuarios y desarrolladores.

Primero se describen los actores que se identificaron y después los subsistemas que forman el proyecto.

Se identificaron 8 subsistemas en el proyecto de los que los cuatro primeros y el de resultados proporcionan acciones similares a los actores. Se detallarán a continuación:

2.2.1. Actores

Los actores son las entidades que interactuarán con la plataforma. Son los de las siguientes tablas:

ID	**ACT-1**
Nombre	Usuario.
Descripción	Un usuario de la interfaz que busca usar la aplicación. Es el foco de la interfaz y en general de la aplicación.

Tabla 2.2: Actor ACT-1. Usuario.

ID	ACT-2
Nombre	Administrador.
Descripción	El administrador es un usuario (tiene todas las capacidades del mismo), con las diferencias de que es el que configura el servidor (es capaz de configurar un cluster, cambiar el tiempo de cuentas de invitado o desactivarlas...) y tiene permisos especiales como el borrado y actualización de todos los usuarios, etc.

Tabla 2.4: Actor ACT-2. Administrador.

ID	ACT-3
Nombre	Desarrollador.
Descripción	Un desarrollador que busca usar el programa como librería, utilizando las distintas funciones de las que dispone: carga y exportación de registros, consultas, restricciones, resultados... y ejecución del algoritmo. Este desarrollador buscará integrar el sistema con otros programas que esté usando.

Tabla 2.6: Actor ACT-3. Desarrollador.

ID	ACT-4
Nombre	Desarrollador web.
Descripción	Un desarrollador que busca usar el programa como librería, siempre a través del servidor desarrollado mediante peticiones web directas. Probablemente busque automatizar ciertas tareas o escribir su propia interfaz para el servidor, por ejemplo para una sistema embebido sin navegador que no puede mostrar la interfaz actual.

Tabla 2.8: Actor ACT-4. Desarrollador web.

2.2.2. Subsistema 1: Registros

Este subsistema se encarga del manejo de ficheros de registro, su carga en memoria y su procesado, por ejemplo obteniendo la lista de actividades que contiene.

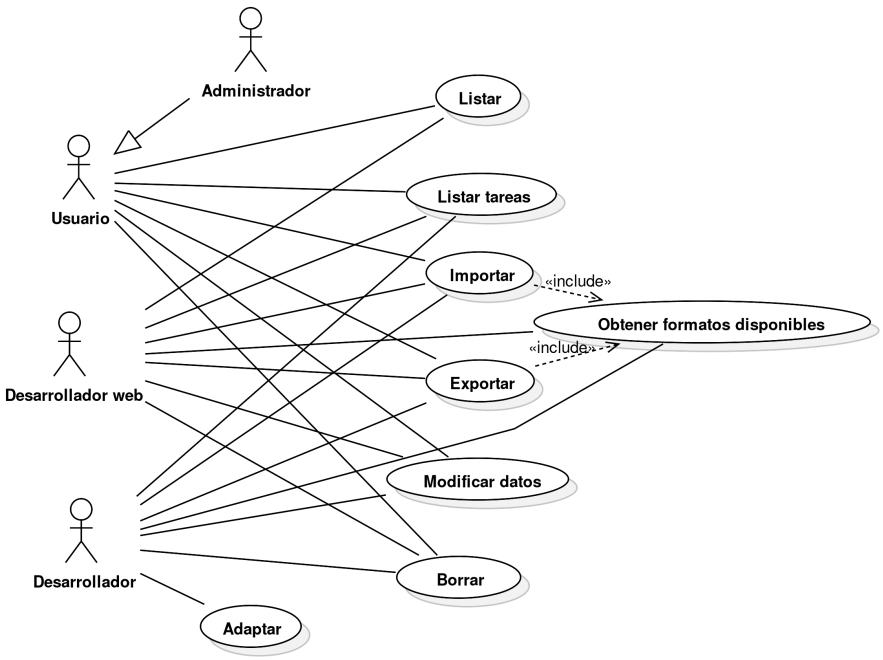


Figura 2.2: Subsistema 1: Registros

Después se describen los casos de uso. No se incluyen el campo casos de uso relacionados porque son casos de uso simples y se pueden ver en el propio diagrama sin necesidad de duplicar la información. El diagrama se puede ver en la Figura 2.2 y la especificación de los casos de uso comienza a continuación.

ID	**CU-1**
Nombre	*Registros:* Listar.
Descripción	Listar los registros.
Actor	Actor ACT-0. Usuario., Actor ACT-0. Administrador., Actor ACT-0. Desarrollador web..
Precondición	El usuario ha iniciado sesión.
Secuencia normal	1. El usuario solicita la lista de registros
Escenario alternativo 1	1. El usuario aplica un conjunto de filtros a campos de forma previa a solicitar la lista.
Escenario alternativo 2	1. El usuario ordena, cambia de página o su tamaño de forma previa a solicitar la lista.
Postcondición	El usuario dispone de la lista de registros solicitada.
Frecuencia	Elevada.

Tabla 2.10: Caso de uso CU-1. *Registros:* Listar.

ID	CU-2
Nombre	*Registros:* Listar tareas.
Descripción	Listar actividades de un registro.
Actor	Actor ACT-0. Usuario., Actor ACT-0. Administrador., Actor ACT-0. Desarrollador., Actor ACT-0. Desarrollador web..
Precondición	El usuario ha iniciado sesión y seleccionado un registro.
Secuencia normal	1. El usuario pulsa en editar una consulta del registro
Postcondición	El usuario dispone de la lista de actividades del registro solicitado para ayudar con la edición de la consulta en la que deben aparecer las actividades.
Frecuencia	Muy elevada.

Tabla 2.12: Caso de uso CU-2. *Registros:* Listar tareas.

ID	CU-3
Nombre	*Registros:* Importar.
Descripción	Importar un registro desde un fichero que puede ser en formato CSV, XES u otros.
Actor	Actor ACT-0. Usuario., Actor ACT-0. Administrador., Actor ACT-0. Desarrollador., Actor ACT-0. Desarrollador web..
Precondición	El usuario ha iniciado sesión.
Secuencia normal	1. El usuario pulsa en importar. 2. El usuario introduce los metadatos. 3. **Inclusión**: Caso de uso CU-3. *Registros:* Obtener formatos disponibles.. 4. El usuario selecciona el fichero.
Escenario alternativo 1	4. El usuario configura la importación del fichero si según el formato es necesario y vuelve al paso 4.
Escenario alternativo 2	4. El usuario recibe el error de formato no soportado.
Postcondición	El usuario puede ver que el registro aparece en el listado.
Frecuencia	Elevada.

Tabla 2.14: Caso de uso CU-3. *Registros:* Importar.

ID	CU-4
Nombre	*Registros:* Obtener formatos disponibles.
Descripción	Obtener los formatos disponibles para la importación y exportación de un registro, y la posible configuración.
Actor	Actor ACT-0. Desarrollador., Actor ACT-0. Desarrollador web..
Precondición	El usuario ha iniciado sesión y seleccionado un registro.
Secuencia normal	1. El usuario pulsa en importar/exportar el registro
Postcondición	Se obtienen los formatos disponibles.
Frecuencia	Elevada.

Tabla 2.16: Caso de uso CU-4. *Registros:* Obtener formatos disponibles.

ID	CU-5
Nombre	*Registros:* Exportar.
Descripción	Exportar un registro a un fichero que puede ser en formato CSV, XES u otros.
Actor	Actor ACT-0. Usuario., Actor ACT-0. Administrador., Actor ACT-0. Desarrollador., Actor ACT-0. Desarrollador web..
Precondición	El usuario ha iniciado sesión.
Secuencia normal	1. El usuario pulsa en exportar. 2. **Inclusión**: Caso de uso CU-5. *Registros:* Obtener formatos disponibles.. 3. El usuario selecciona el formato.
Escenario alternativo 1	2. El usuario configura la exportación del fichero si según el formato es necesario y vuelve al paso 2.
Postcondición	El usuario completa la descarga.
Frecuencia	Baja.

Tabla 2.18: Caso de uso CU-5. *Registros:* Exportar.

ID	CU-6
Nombre	*Registros:* Modificar datos.
Descripción	Modificar los datos asociados de los elementos que pertenezcan al usuario.
Actor	Actor ACT-0. Usuario., Actor ACT-0. Administrador., Actor ACT-0. Desarrollador., Actor ACT-0. Desarrollador web..
Precondición	El usuario ha iniciado sesión.
Secuencia normal	1. El usuario selecciona editar en el registro que desea. 2. El usuario introduce y confirma los metadatos.
Postcondición	El usuario ve los nuevos metadatos.
Frecuencia	Media.

Tabla 2.20: Caso de uso CU-6. *Registros:* Modificar datos.

ID	CU-7
Nombre	*Registros:* Borrar.
Descripción	Borrar los datos asociados de un registro que pertenezca al usuario.
Actor	Actor ACT-0. Usuario., Actor ACT-0. Administrador., Actor ACT-0. Desarrollador., Actor ACT-0. Desarrollador web..
Precondición	El usuario ha iniciado sesión.
Secuencia normal	1. El usuario selecciona borrar en el registro que desea. 2. El usuario pulsa confirmar.
Postcondición	El usuario no ve el registro en la lista.
Frecuencia	Media.

Tabla 2.22: Caso de uso CU-7. *Registros:* Borrar.

ID	CU-8
Nombre	*Registros:* Adaptar.
Descripción	Adaptar los registros cargados en Java de otro sistema a este.
Actor	Actor ACT-0. Desarrollador.
Precondición	El desarrollador tiene acceso al proyecto compilado.
Secuencia normal	1. El desarrollador busca la clase del paquete adaptadores adecuada. 2. El desarrollador la llama para adaptar su modelo, y opcionalmente conecta un método que escuche el progreso.
Postcondición	El desarrollador dispone del modelo adaptado.
Frecuencia	Baja.

Tabla 2.24: Caso de uso CU-8. *Registros:* Adaptar.

2.2.3. Subsistema 2: Consultas

Este subsistema se encarga del manejo de consultas, refiriéndose a los procesos que define el usuario para definir el modelo que busca, sin incluir el manejo de restricciones aplicadas sobre el mismo. Se puede ver en la Figura 2.3. La especificación de los casos de uso comienza a continuación.

ID	CU-9
Nombre	*Consultas:* Crear.
Descripción	Crear una consulta a partir de código fuente.
Actor	Actor ACT-0. Usuario., Actor ACT-0. Administrador., Actor ACT-0. Desarrollador., Actor ACT-0. Desarrollador web..
Precondición	El usuario ha iniciado sesión y seleccionado un registro.
Secuencia normal	1. El usuario pulsa en crear consulta 2. El usuario inserta el código fuente y la guarda.
Postcondición	El usuario ve la consulta creada asociada al log.
Frecuencia	Elevada.

Tabla 2.26: Caso de uso CU-9. *Consultas:* Crear.

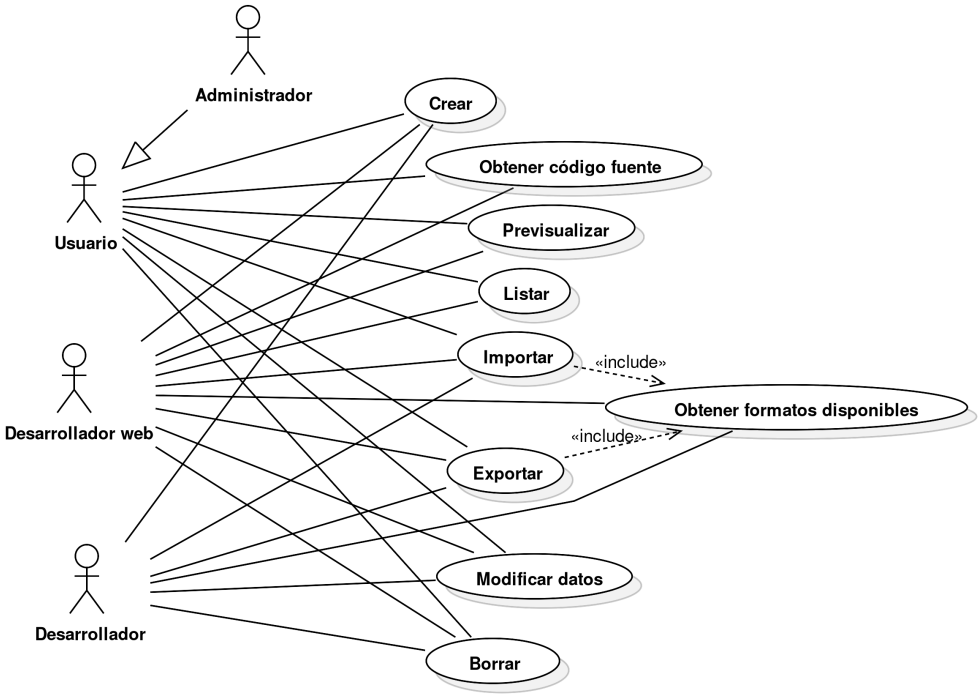


Figura 2.3: Subsistema 2: Consultas

ID	**CU-10**
Nombre	*Consultas:* Obtener código fuente.
Descripción	El usuario obtiene el código fuente de una consulta.
Actor	Actor ACT-0. Usuario., Actor ACT-0. Administrador., Actor ACT-0. Desarrollador web..
Precondición	El usuario ha iniciado sesión y seleccionado una consulta.
Secuencia normal	1. El usuario pulsa en editar consulta
Postcondición	El usuario ve la consulta con el código fuente.
Frecuencia	Elevada.

Tabla 2.28: Caso de uso CU-10. *Consultas:* Obtener código fuente.

ID	CU-11
Nombre	*Consultas:* Previsualizar.
Descripción	Previsualizar una consulta.
Actor	Actor ACT-0. Usuario., Actor ACT-0. Administrador., Actor ACT-0. Desarrollador., Actor ACT-0. Desarrollador web..
Precondición	El usuario ha iniciado sesión y seleccionado un registro.
Secuencia normal	1. El usuario pulsa en ver o editar una consulta.
Postcondición	El usuario puede previsualizar la consulta como grafo y como árbol de análisis sintáctico editable.
Frecuencia	Muy elevada.

Tabla 2.30: Caso de uso CU-11. *Consultas:* Previsualizar.

ID	CU-12
Nombre	*Consultas:* Listar.
Descripción	Listar las consultas.
Actor	Actor ACT-0. Usuario., Actor ACT-0. Administrador., Actor ACT-0. Desarrollador web..
Precondición	El usuario ha iniciado sesión y seleccionado un registro.
Secuencia normal	1. El usuario solicita la lista de consultas
Escenario alternativo 1	1. El usuario aplica un conjunto de filtros.
Escenario alternativo 2	1. El usuario ordena, cambia de página o su tamaño.
Postcondición	El usuario dispone de la lista de consultas solicitada.
Frecuencia	Muy elevada.

Tabla 2.32: Caso de uso CU-12. *Consultas:* Listar.

ID	CU-13
Nombre	*Consultas:* Importar.
Descripción	Importar una consulta desde un fichero que puede ser en formato HN, CN...
Actor	Actor ACT-0. Usuario., Actor ACT-0. Administrador., Actor ACT-0. Desarrollador., Actor ACT-0. Desarrollador web..
Precondición	El usuario ha iniciado sesión y está editando una consulta.
Secuencia normal	1. El usuario pulsa en importar consulta. 2. **Inclusión**: Caso de uso CU-13. *Consultas:* Obtener formatos disponibles.. 3. El usuario selecciona el fichero.
Escenario alternativo 1	2. El usuario configura la importación del fichero si según el formato es necesario y vuelve al paso 2.
Escenario alternativo 2	2. El usuario recibe el error de formato no soportado.
Postcondición	El usuario puede ver que la consulta aparece en el listado.
Frecuencia	Elevada.

Tabla 2.34: Caso de uso CU-13. *Consultas:* Importar.

ID	CU-14
Nombre	*Consultas:* Obtener formatos disponibles.
Descripción	Obtener los formatos disponibles para la importación y exportación de consultas, y la posible configuración.
Actor	Actor ACT-0. Desarrollador., Actor ACT-0. Desarrollador web..
Precondición	El usuario ha iniciado sesión.
Secuencia normal	1. El usuario pulsa en importar/exportar consulta.
Postcondición	Se obtienen los formatos disponibles.
Frecuencia	Elevada.

Tabla 2.36: Caso de uso CU-14. *Consultas:* Obtener formatos disponibles.

ID	CU-15
Nombre	*Consultas:* Exportar.
Descripción	Exportar una consulta a un fichero que puede ser en formato HN, CN u otros.
Actor	Actor ACT-0. Usuario., Actor ACT-0. Administrador., Actor ACT-0. Desarrollador., Actor ACT-0. Desarrollador web..
Precondición	El usuario ha iniciado sesión y está viendo una consulta.
Secuencia normal	1. El usuario pulsa en exportar. 2. **Inclusión**: Caso de uso CU-15. *Consultas:* Obtener formatos disponibles.. 3. El usuario selecciona el formato.
Escenario alternativo 1	2. El usuario configura la exportación del fichero si según el formato es necesario y vuelve al paso 2.
Postcondición	El usuario completa la descarga.
Frecuencia	Baja.

Tabla 2.38: Caso de uso CU-15. *Consultas:* Exportar.

ID	CU-16
Nombre	*Consultas:* Modificar datos.
Descripción	Modificar los metadatos asociados de los elementos que pertenezcan al usuario.
Actor	Actor ACT-0. Usuario., Actor ACT-0. Administrador., Actor ACT-0. Desarrollador., Actor ACT-0. Desarrollador web..
Precondición	El usuario ha iniciado sesión y viendo una registro.
Secuencia normal	1. El usuario selecciona editar en la consulta que desea. 2. El usuario introduce y confirma los metadatos.
Postcondición	El usuario ve los nuevos metadatos.
Frecuencia	Media.

Tabla 2.40: Caso de uso CU-16. *Consultas:* Modificar datos.

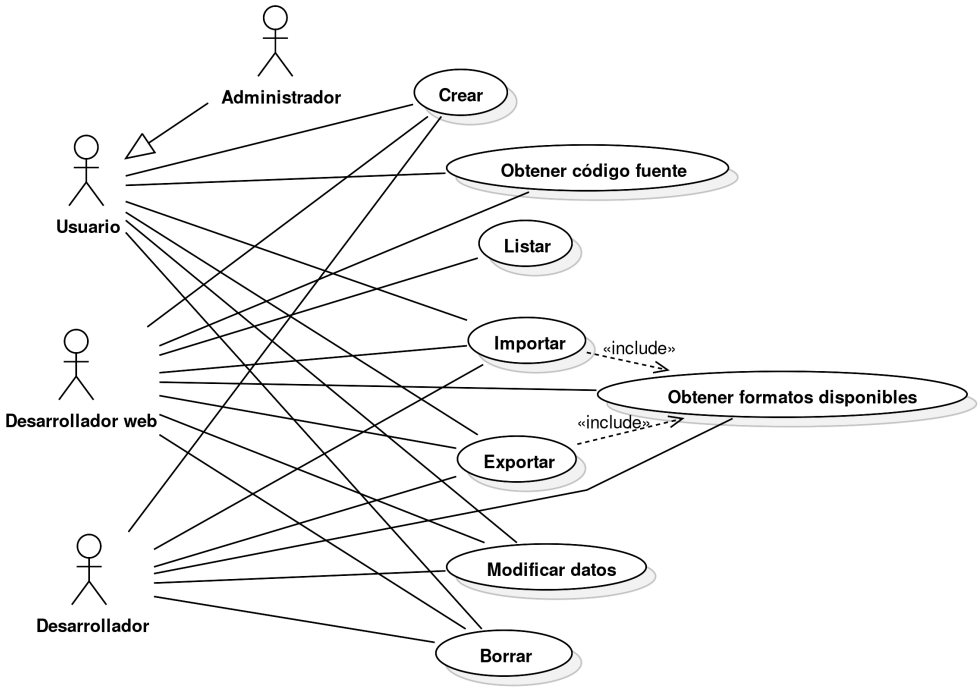


Figura 2.4: Subsistema 3: Restricciones

ID	CU-17
Nombre	*Consultas:* Borrar.
Descripción	Borrar los datos asociados a una consulta que pertenezca al usuario.
Actor	Actor ACT-0. Usuario., Actor ACT-0. Administrador., Actor ACT-0. Desarrollador., Actor ACT-0. Desarrollador web..
Precondición	El usuario ha iniciado sesión y está viendo un registro.
Secuencia normal	1. El usuario selecciona borrar en la consulta que desea. 2. El usuario pulsa confirmar.
Postcondición	El usuario no ve la consulta en la lista.
Frecuencia	Media.

Tabla 2.42: Caso de uso CU-17. *Consultas:* Borrar.

2.2.4. Subsistema 3: Restricciones

Este subsistema se encarga del manejo de restricciones que se aplicarán sobre las consultas anteriores. Se puede ver en la Figura 2.4. La especificación de los casos de uso comienza a continuación.

ID	CU-18
Nombre	*Restricciones:* Crear.
Descripción	Crear una restricción a partir de código fuente.
Actor	Actor ACT-0. Usuario., Actor ACT-0. Administrador., Actor ACT-0. Desarrollador., Actor ACT-0. Desarrollador web..
Precondición	El usuario ha iniciado sesión y seleccionado un registro.
Secuencia normal	1. El usuario pulsa en crear o editar consulta. 2. El usuario inserta el código fuente de la restricción.
Postcondición	El usuario ve la restricción creada asociada a la consulta.
Frecuencia	Elevada.

Tabla 2.44: Caso de uso CU-18. *Restricciones:* Crear.

ID	CU-19
Nombre	*Restricciones:* Obtener código fuente.
Descripción	El usuario obtiene el código fuente de una restricción.
Actor	Actor ACT-0. Usuario., Actor ACT-0. Administrador., Actor ACT-0. Desarrollador web..
Precondición	El usuario ha iniciado sesión y seleccionado una consulta.
Secuencia normal	1. El usuario pulsa en editar consulta
Postcondición	El usuario ve la restricción con el código fuente.
Frecuencia	Elevada.

Tabla 2.46: Caso de uso CU-19. *Restricciones:* Obtener código fuente.

ID	CU-20
Nombre	*Restricciones:* Listar.
Descripción	Listar las restricciones.
Actor	Actor ACT-0. Usuario., Actor ACT-0. Administrador., Actor ACT-0. Desarrollador web..
Precondición	El usuario ha iniciado sesión y seleccionado un registro.
Secuencia normal	1. El usuario solicita la lista de restricciones
Escenario alternativo 1	1. El usuario aplica un conjunto de filtros.
Escenario alternativo 2	1. El usuario ordena, cambia de página o su tamaño.
Postcondición	El usuario dispone de la lista de restricciones solicitada.
Frecuencia	Muy elevada.

Tabla 2.48: Caso de uso CU-20. *Restricciones:* Listar.

ID	CU-21
Nombre	*Restricciones:* Importar.
Descripción	Importar una restricción desde un fichero.
Actor	Actor ACT-0. Usuario., Actor ACT-0. Administrador., Actor ACT-0. Desarrollador., Actor ACT-0. Desarrollador web..
Precondición	El usuario ha iniciado sesión y está editando una consulta.
Secuencia normal	1. El usuario pulsa en importar restricción. 2. **Inclusión**: Caso de uso CU-21. *Restricciones:* Obtener formatos disponibles.. 3. El usuario selecciona el fichero.
Escenario alternativo 1	2. El usuario configura la importación del fichero si según el formato es necesario y vuelve al paso 2.
Escenario alternativo 2	2. El usuario recibe el error de formato no soportado.
Postcondición	El usuario puede ver que la restricción aparece en el listado.
Frecuencia	Elevada.

Tabla 2.50: Caso de uso CU-21. *Restricciones:* Importar.

ID	CU-22
Nombre	*Restricciones:* Obtener formatos disponibles.
Descripción	Obtener los formatos disponibles para la importación y exportación de restricciones, y la posible configuración.
Actor	Actor ACT-0. Desarrollador., Actor ACT-0. Desarrollador web..
Precondición	El usuario ha iniciado sesión.
Secuencia normal	1. El usuario pulsa en importar/exportar restricción.
Postcondición	Se obtienen los formatos disponibles.
Frecuencia	Elevada.

Tabla 2.52: Caso de uso CU-22. *Restricciones:* Obtener formatos disponibles.

ID	CU-23
Nombre	*Restricciones:* Exportar.
Descripción	Exportar una restricción a un fichero que puede ser en formato HN, CN u otros.
Actor	Actor ACT-0. Usuario., Actor ACT-0. Administrador., Actor ACT-0. Desarrollador., Actor ACT-0. Desarrollador web..
Precondición	El usuario ha iniciado sesión y está viendo una restricción.
Secuencia normal	1. El usuario pulsa en exportar. 2. **Inclusión**: Caso de uso CU-23. *Consultas:* Obtener formatos disponibles.. 3. El usuario selecciona el formato.
Escenario alternativo 1	2. El usuario configura la exportación del fichero si según el formato es necesario y vuelve al paso 2.
Postcondición	El usuario completa la descarga.
Frecuencia	Baja.

Tabla 2.54: Caso de uso CU-23. *Restricciones:* Exportar.

ID	CU-24
Nombre	*Restricciones:* Modificar datos.
Descripción	Modificar los metadatos asociados de los elementos que pertenezcan al usuario.
Actor	Actor ACT-0. Usuario., Actor ACT-0. Administrador., Actor ACT-0. Desarrollador., Actor ACT-0. Desarrollador web..
Precondición	El usuario ha iniciado sesión y viendo una registro.
Secuencia normal	1. El usuario selecciona editar en la restricción que desea. 2. El usuario introduce y confirma los metadatos.
Postcondición	El usuario ve los nuevos metadatos.
Frecuencia	Media.

Tabla 2.56: Caso de uso CU-24. *Restricciones:* Modificar datos.

ID	CU-25
Nombre	*Restricciones:* Borrar.
Descripción	Borrar los datos asociados a una restricción que pertenezca al usuario.
Actor	Actor ACT-0. Usuario., Actor ACT-0. Administrador., Actor ACT-0. Desarrollador., Actor ACT-0. Desarrollador web..
Precondición	El usuario ha iniciado sesión y está viendo una registro.
Secuencia normal	1. El usuario selecciona borrar en la restricción que desea. 2. El usuario pulsa confirmar.
Postcondición	El usuario no ve la restricción en la lista.
Frecuencia	Media.

Tabla 2.58: Caso de uso CU-25. *Restricciones:* Borrar.

2.2.5. Subsistema 4: Configuración

Este subsistema controla las configuración que hace el usuario antes de ejecutar cada algoritmo, lo cual incluye los registros, restricciones y restricciones usados, un nombre y descripción, el *fitness* mínimo y si se aplican otros añadidos al algoritmo, como el cálculo medio del tiempo entre tareas. Se puede ver en la Figura 2.5. La especificación de los casos de uso comienza a continuación.

Este subsistema sufrió cambios a la hora de mostrar el resultado del tercer incremento a los tutores. Como se explicará en ese incremento, los tutores indicaron que sería recomendable una simplificación de este sistema que antes era

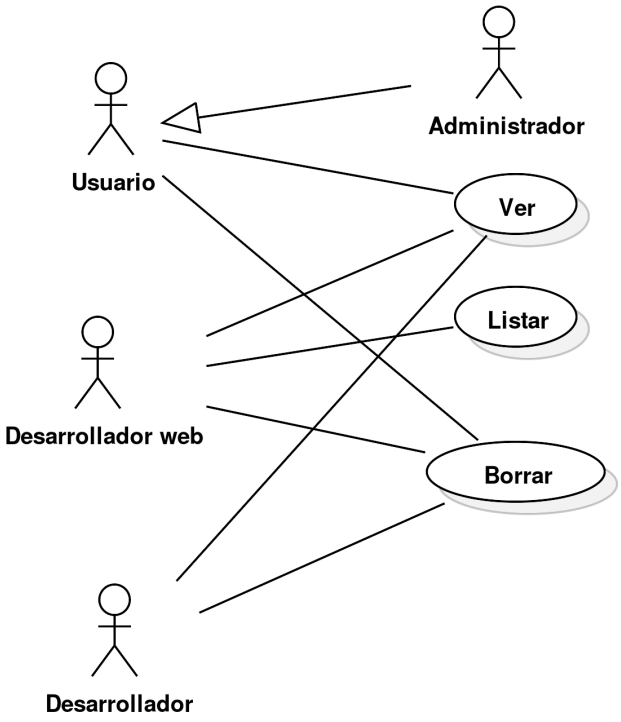


Figura 2.5: Subsistema 4: Configuración

similar a los subsistemas anteriores (creación, modificación...). La implementación del servidor sigue aceptando todos estos casos de uso, pero en la interfaz estas acciones se escondieron bajo el interruptor que indica si se usa la nueva interfaz o la anterior, por lo que también se simplificaron los casos de uso, aunque sigan implementados.

ID	**CU-26**
Nombre	*Configuración:* Ver.
Descripción	Ver la configuración asociada a un resultado.
Actor	Actor ACT-0. Usuario., Actor ACT-0. Administrador., Actor ACT-0. Desarrollador., Actor ACT-0. Desarrollador web..
Precondición	El usuario ha iniciado sesión y seleccionado un resultado.
Secuencia normal	1. El usuario baja para ver la configuración.
Postcondición	El usuario puede visualizar toda la configuración usada para la ejecución que está viendo.
Frecuencia	Elevada.

Tabla 2.60: Caso de uso CU-26. *Configuración:* Ver.

ID	CU-27
Nombre	*Configuración:* Listar.
Descripción	Listar las configuraciones de los resultados de una consulta.
Actor	Actor ACT-0. Desarrollador web..
Precondición	El desarrollador ha iniciado sesión y seleccionado una consulta.
Secuencia normal	1. El desarrollador solicita la lista de configuraciones.
Escenario alternativo 1	1. El desarrollador aplica un conjunto de filtros.
Escenario alternativo 2	1. El desarrollador ordena, cambia de página o su tamaño.
Postcondición	El desarrollador dispone de la lista de configuraciones solicitada.
Frecuencia	Muy elevada.

Tabla 2.62: Caso de uso CU-27. *Configuración:* Listar.

ID	CU-28
Nombre	*Configuración:* Borrar.
Descripción	Borrar los datos asociados a una configuración y su resultado que pertenezca al usuario.
Actor	Actor ACT-0. Usuario., Actor ACT-0. Administrador., Actor ACT-0. Desarrollador., Actor ACT-0. Desarrollador web..
Precondición	El usuario ha iniciado sesión y está viendo una consulta.
Secuencia normal	1. El usuario selecciona borrar en la configuración que desea. 2. El usuario pulsa confirmar.
Postcondición	El usuario no ve la configuración en la lista.
Frecuencia	Media.

Tabla 2.64: Caso de uso CU-28. *Configuración:* Borrar.

2.2.6. Subsistema 5: Ejecución

Este subsistema es el que permite a los actores la ejecución del algoritmo. Se puede ver en la Figura 2.6. La especificación de los casos de uso comienza a continuación.

Los dos primeros actores (usuario y desarrollador web) no tienen elección entre local o remoto, sino que lo decide el administrador del sistema si configura

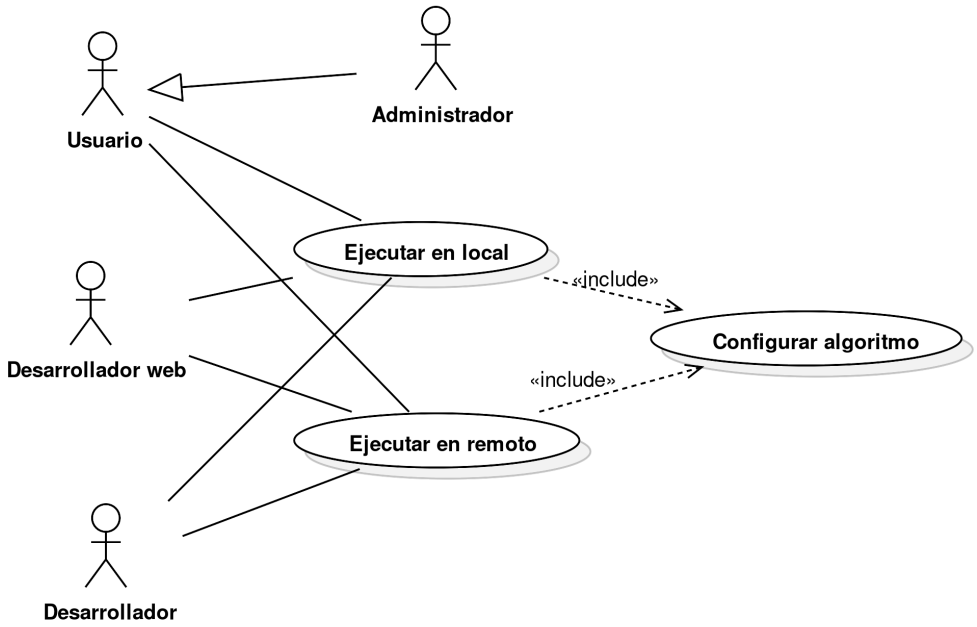


Figura 2.6: Subsistema 5: Ejecución

el clúster remoto. Tienen la opción de desplegar su propio servidor (simplemente ejecutar un JAR) configurando una instancia de un clúster del que sepan la contraseña como sistema remoto.

Desde el punto de vista de los usuarios no existe diferencia entre la ejecución local o remota, excepto para el actor desarrollador, que es capaz de mejorar el rendimiento de la aplicación si realiza pequeños cambios o diseña su código para soportar el trabajo remoto. Por este motivo se incluye como caso de uso específico para el desarrollador del que también se benefician los otros actores.

Las funcionalidades extra del algoritmo se pueden configurar porque añaden un coste (principalmente de sincronización de datos entre nodos de un clúster necesaria), como calcular el tiempo medio entre las tareas.

ID	CU-29
Nombre	*Ejecución:* Ejecutar en local.
Descripción	Ejecutar el algoritmo sobre registro y consulta (con o sin restricción) en el propio servidor que presenta la interfaz.
Actor	Actor ACT-0. Usuario., Actor ACT-0. Administrador., Actor ACT-0. Desarrollador., Actor ACT-0. Desarrollador web..
Precondición	El usuario ha iniciado sesión y está viendo una consulta.
Secuencia normal	1. El usuario pulsa ejecutar. 2. **Inclusión**: Caso de uso CU-29. *Ejecución:* Configurar algoritmo.. 3. El usuario espera mientras ve el progreso de la ejecución.
Postcondición	El usuario puede ver los resultados.
Frecuencia	Elevada.

Tabla 2.66: Caso de uso CU-29. *Ejecución:* Ejecutar en local.

ID	CU-30
Nombre	*Ejecución:* Ejecutar en remoto.
Descripción	Ejecutar el algoritmo sobre registro y consulta (con o sin restricción) en un servidor o clúster Spark externo de mayor eficiencia.
Actor	Actor ACT-0. Usuario., Actor ACT-0. Administrador., Actor ACT-0. Desarrollador., Actor ACT-0. Desarrollador web..
Precondición	El usuario ha iniciado sesión y está viendo una consulta.
Secuencia normal	1. El usuario pulsa ejecutar. 2. **Inclusión**: Caso de uso CU-30. *Ejecución:* Configurar algoritmo.. 3. El usuario espera mientras ve el progreso de la ejecución.
Postcondición	El usuario puede ver los resultados.
Frecuencia	Elevada.

Tabla 2.68: Caso de uso CU-30. *Ejecución:* Ejecutar en remoto.

ID	CU-31
Nombre	*Ejecución:* Configurar algoritmo.
Descripción	Proporcionar opciones de configuración al algoritmo de forma previa a la ejecución.
Actor	Actor ACT-0. Usuario., Actor ACT-0. Administrador., Actor ACT-0. Desarrollador., Actor ACT-0. Desarrollador web..
Precondición	El usuario ha pulsado en ejecutar el algoritmo.
Secuencia normal	1. El usuario da nombre y descripción a la configuración. 2. El usuario configura el fitness mínimo. 3. El usuario escoge las funcionalidades extra. 4. El usuario confirma la configuración.
Postcondición	El usuario ha iniciado la ejecución con esta configuración.
Frecuencia	Elevada.

Tabla 2.70: Caso de uso CU-31. *Ejecución:* Configurar algoritmo.

2.2.7. Subsistema 6: Resultados

Este subsistema permite visualizar y exportar los resultados. Los resultados se generan automáticamente al ejecutar el algoritmo y no tiene sentido dar la posibilidad de modificarlos, excepto el desarrollador que tiene esa opción. Se puede ver en la Figura 2.7. La especificación de los casos de uso comienza a continuación.

ID	CU-32
Nombre	*Resultados:* Listar.
Descripción	Listar los resultados.
Actor	Actor ACT-0. Desarrollador web..
Precondición	El usuario ha iniciado sesión y seleccionado una consulta.
Secuencia normal	1. El usuario solicita la lista de resultados
Escenario alternativo 1	1. El usuario aplica un conjunto de filtros.
Escenario alternativo 2	1. El usuario ordena, cambia de página o su tamaño.
Postcondición	El usuario dispone de la lista de resultados solicitada.
Frecuencia	Muy elevada.

Tabla 2.72: Caso de uso CU-32. *Resultados:* Listar.

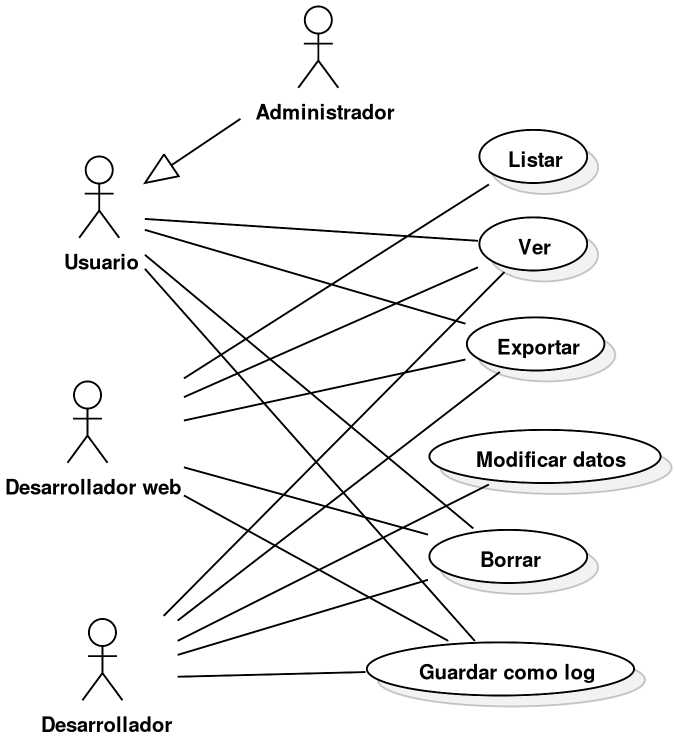


Figura 2.7: Subsistema 6: Resultados

ID	**CU-33**
Nombre	*Resultados:* Ver.
Descripción	Ver los datos de un resultado.
Actor	Actor ACT-0. Usuario., Actor ACT-0. Administrador., Actor ACT-0. Desarrollador., Actor ACT-0. Desarrollador web..
Precondición	El usuario ha iniciado sesión y está viendo un registro.
Secuencia normal	1. El usuario va a la consulta que ha ejecutado. 2. El usuario pulsa en la ejecución realizada.
Postcondición	El usuario puede ver el resultado y otros datos asociados.
Frecuencia	Elevada.

Tabla 2.74: Caso de uso CU-33. *Resultados:* Ver.

ID	CU-34
Nombre	*Resultados:* Exportar.
Descripción	Exportar un resultado a un fichero que puede ser en formato HN, CN u otros.
Actor	Actor ACT-0. Usuario., Actor ACT-0. Administrador., Actor ACT-0. Desarrollador., Actor ACT-0. Desarrollador web..
Precondición	El usuario ha iniciado sesión y está viendo un resultado.
Secuencia normal	1. El usuario pulsa en exportar.
Escenario alternativo 1	2. El usuario configura la exportación del fichero si según el formato es necesario y vuelve al paso 2.
Postcondición	El usuario completa la descarga.
Frecuencia	Baja.

Tabla 2.76: Caso de uso CU-34. *Resultados:* Exportar.

ID	CU-35
Nombre	*Resultados:* Modificar datos.
Descripción	Modificar los datos asociados de los resultados.
Actor	Actor ACT-0. Desarrollador..
Precondición	El desarrollador dispone de una consulta con resultados.
Secuencia normal	1. El desarrollador edita los datos.
Postcondición	El desarrollador ve los nuevos datos.
Frecuencia	Baja.

Tabla 2.78: Caso de uso CU-35. *Resultados:* Modificar datos.

ID	CU-36
Nombre	*Resultados:* Borrar.
Descripción	Borrar los datos asociados a un resultado que pertenezca al usuario.
Actor	Actor ACT-0. Usuario., Actor ACT-0. Administrador., Actor ACT-0. Desarrollador., Actor ACT-0. Desarrollador web..
Precondición	El usuario ha iniciado sesión y está viendo una consulta.
Secuencia normal	1. El usuario selecciona borrar en el resultado que desea. 2. El usuario pulsa confirmar.
Postcondición	El usuario no ve el resultado en la lista.
Frecuencia	Media.

Tabla 2.80: Caso de uso CU-36. *Resultados:* Borrar.

ID	CU-37
Nombre	*Resultados:* Guardar como log.
Descripción	Guarda las trazas del log que son aceptadas por el modelo y las restricciones en este resultado como un nuevo resultado.
Actor	Actor ACT-0. Usuario., Actor ACT-0. Administrador., Actor ACT-0. Desarrollador., Actor ACT-0. Desarrollador web..
Precondición	El usuario ha iniciado sesión y ha configurado el algoritmo para guardar las trazas validas en la ejecución que ha terminado.
Secuencia normal	1. El usuario accede al resultado. 2. El usuario introduce el nombre del nuevo log. 3. El usuario pulsa guardar como nuevo log.
Postcondición	El usuario ve el nuevo log en la lista.
Frecuencia	Media.

Tabla 2.82: Caso de uso CU-37. *Resultados:* Guardar como log.

2.2.8. Subsistema 7: Usuarios

Este subsistema se encarga del manejo de usuarios. Se puede ver en la Figura 2.8. La especificación de los casos de uso comienza a continuación.

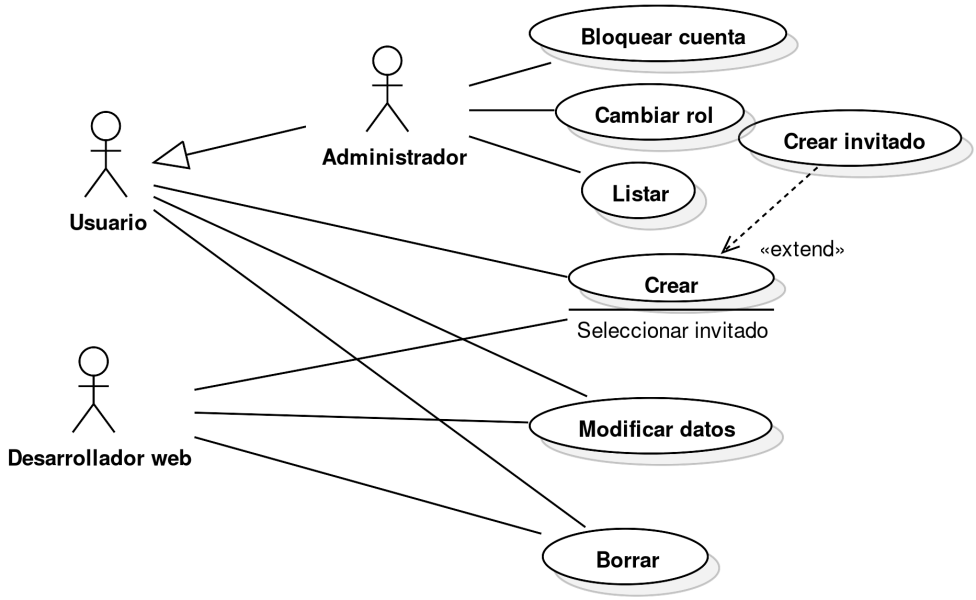


Figura 2.8: Subsistema 7: Usuarios

ID	**CU-38**
Nombre	*Usuarios:* Bloquear cuenta.
Descripción	No se permite el inicio de sesión hasta desbloqueo.
Actor	Actor ACT-0. Administrador..
Precondición	—
Secuencia normal	1. El administrador llama al servicio de bloqueo de cuenta.
Postcondición	El usuario ya no puede iniciar sesión (o puede).
Frecuencia	Baja.

Tabla 2.84: Caso de uso CU-38. *Usuarios:* Bloquear cuenta.

ID	CU-39
Nombre	*Usuarios:* Cambiar rol.
Descripción	Cambiar el rol de un usuario.
Actor	Actor ACT-0. Administrador..
Precondición	—
Secuencia normal	1. El administrador llama al servicio de cambio de rol.
Postcondición	El usuario tiene el nuevo rol.
Frecuencia	Baja.

Tabla 2.86: Caso de uso CU-39. *Usuarios:* Cambiar rol.

ID	CU-40
Nombre	*Usuarios:* Listar.
Descripción	Listar los usuarios registrados.
Actor	Actor ACT-0. Administrador..
Precondición	—
Secuencia normal	1. El administrador llama al servicio de listar usuarios.
Postcondición	El administrador recibe el listado de usuarios.
Frecuencia	Baja.

Tabla 2.88: Caso de uso CU-40. *Usuarios:* Listar.

ID	CU-41
Nombre	*Usuarios:* Crear.
Descripción	Crear un usuario con mínimos privilegios.
Actor	Actor ACT-0. Usuario., Actor ACT-0. Administrador., Actor ACT-0. Desarrollador..
Precondición	El usuario se ha conectado a la interfaz.
Secuencia normal	1. **Extensión**: Caso de uso CU-41. *Usuarios:* Crear invitado. 2. El usuario inserta datos y crea la cuenta.
Postcondición	El usuario puede iniciar sesión.
Frecuencia	Elevada.

Tabla 2.90: Caso de uso CU-41. *Usuarios:* Crear.

ID	CU-42
Nombre	*Usuarios:* Crear invitado.
Descripción	Crear un usuario que será borrado, con todos sus datos.
Actor	Actor ACT-0. Usuario., Actor ACT-0. Administrador., Actor ACT-0. Desarrollador..
Precondición	El usuario se ha conectado a la interfaz.
Secuencia normal	1. El usuario crea invitado y se inicia sesión.
Postcondición	El usuario tiene una cuenta de duración limitada.
Frecuencia	Elevada.

Tabla 2.92: Caso de uso CU-42. *Usuarios:* Crear invitado.

ID	CU-43
Nombre	*Usuarios:* Modificar datos.
Descripción	Modificar los datos asociados de los resultados.
Actor	Actor ACT-0. Desarrollador..
Precondición	El desarrollador dispone de una consulta con resultados.
Secuencia normal	1. El desarrollador edita los datos.
Postcondición	El desarrollador ve los nuevos datos.
Frecuencia	Baja.

Tabla 2.94: Caso de uso CU-43. *Usuarios:* Modificar datos.

ID	CU-44
Nombre	*Usuarios:* Borrar.
Descripción	Borrar los datos asociados a un usuario.
Actor	Actor ACT-0. Usuario., Actor ACT-0. Administrador., Actor ACT-0. Desarrollador..
Precondición	El usuario ha iniciado sesión y está viendo una consulta.
Secuencia normal	1. El usuario realiza la llamada al servicio de borrado.
Postcondición	El usuario no ve puede iniciar sesión.
Frecuencia	Media.

Tabla 2.96: Caso de uso CU-44. *Usuarios:* Borrar.

2.2.9. Subsistema 8: Tareas

Este subsistema permite a los actores recibir información en tiempo real de las tareas que se realizan y su progreso. Las tareas se crean automáticamente con la realización de operaciones largas en el servidor, por lo que al no ser una acción iniciada directamente por un actor no se incluyen en el diagrama. Se

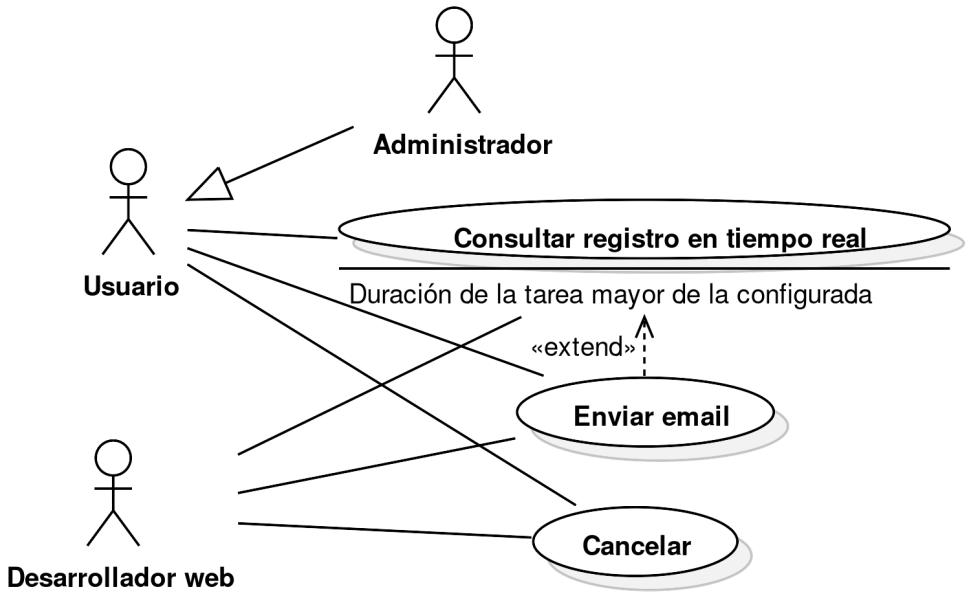


Figura 2.9: Subsistema 8: Tareas

puede ver en la Figura 2.9. La especificación de los casos de uso comienza a continuación.

ID	**CU-45**
Nombre	*Tareas:* Consultar registro en tiempo real.
Descripción	Ver la información que envía el servidor sobre el progreso de las tareas que tardan cierto tiempo, actualizada en tiempo real.
Actor	Actor ACT-0. Usuario., Actor ACT-0. Administrador., Actor ACT-0. Desarrollador web..
Precondición	El usuario ha iniciado sesión.
Secuencia normal	1. El usuario va a la lista de tareas y opcionalmente selecciona una en concreto. 2. El usuario lee el registro y espera actualizaciones.
Postcondición	El usuario ve el progreso y mensajes informativos de la tarea.
Frecuencia	Elevada.

Tabla 2.98: Caso de uso CU-45. *Tareas:* Consultar registro en tiempo real.

ID	CU-46
Nombre	*Tareas:* Enviar email.
Descripción	Recibir un correo con información sobre la tarea una vez finalice si tuvo una duración larga.
Actor	Actor ACT-0. Usuario., Actor ACT-0. Administrador., Actor ACT-0. Desarrollador web..
Precondición	Una tarea iniciada por el usuario tarda bastante (configurable por cada usuario).
Secuencia normal	1. El usuario recibe un correo informando de la finalización de la tarea (exitosa o con error), con un enlace a la interfaz y el registro completo de la propia tarea.
Postcondición	El usuario recibe el nombre, progreso y mensajes informativos de la tarea, además de un enlace a la interfaz.
Frecuencia	Elevada.

Tabla 2.100: Caso de uso CU-46. *Tareas:* Enviar email.

ID	CU-47
Nombre	*Tareas:* Cancelar.
Descripción	Cancelar una tarea en ejecución, que parará lo antes posible, parando la transacción que estaba en progreso.
Actor	Actor ACT-0. Usuario., Actor ACT-0. Administrador., Actor ACT-0. Desarrollador web..
Precondición	Una tarea iniciada por el usuario sigue en ejecución.
Secuencia normal	1. El usuario pulsa cancelar en la tarea indicada.
Postcondición	El usuario recibe la notificación de que se ha cancelado la tarea.
Frecuencia	Baja.

Tabla 2.102: Caso de uso CU-47. *Tareas:* Cancelar.

2.3. Especificación de requisitos

Se pueden clasificar los requisitos de muchas formas distintas. Además de las usadas, se pueden indicar las reglas de negocio, pero para un proyecto de investigación no se ven demasiado útiles. En este proyecto se definirán los siguientes grupos.

De información. Indican los datos que debe almacenar el sistema y que son necesarios para el la aceptación de los requisitos de otros tipos. Estos se relacionarán con los requisitos funcionales mediante un campo en la especificación.

Funcionales. Un requisito funcional define una característica que un sistema debe ofrecer, definiendo el comportamiento interno del software: cálculos, detalles técnicos, manipulación de datos y otras funcionalidades específicas que muestran cómo los casos de uso serán llevados a la práctica. El uso casos de uso para representar requisitos funcionales es muy común en distintos proyectos, y en este se desarrollará una matriz de trazabilidad entre ambos.

No funcionales. Un requisito no funcional describe unos criterios que pueden usarse para validar la operación de un proyecto en lugar de sus comportamientos específicos, es decir, proponen unos límites de rendimiento, disponibilidad tiempos de respuesta, capacidad de almacenamiento, etc.

2.3.1. Requisitos de información

En esta sección, se descibirán los diferentes tipos de requisitos y restricciones que tiene la parte del sistema que se está diseñando. Especifican los datos que debe almacenar el sistema para el cumplimiento, parcial o total, de requisitos de otros tipos. Se ignoran los campos autores (que siempre es Jacobo), fuentes (que siempre son los tutores) y versión (ver gestión de la configuración).

Los requisitos de información especifican los datos que debe almacenar el sistema para el cumplimiento, parcial o total, de requisitos de otros tipo. La urgencia se valora con vital/importante/quedaría bien y la urgencia en inmediatamente/hay presión/puede esperar.

Los campos de un requisito de información se refieren a los atributos de información que se deben manejar para cada entidad detallada. Cuando se indica una referencia a otro requisito de información se debe guardar como referencia, evitando copias.

Los requisitos de información se especifican a continuación.

ID	**IRQ-1**	
Nombre	Registros.	
Descripción	Cada uno de los registros que los usuarios suben a la aplicación para procesar.	
Dependencias	FRQ-1, FRQ-2, FRQ-3, FRQ-4, FRQ-5, FRQ-11.	
Campos	Propietario, número de trazas y de casos, nombre, descripción, consultas asociadas.	
Tiempo de vida	**Medio** 15 días	**Máximo** ∞
Ocurrencias simultáneas	**Medio** 3	**Máximo** ∞
Importancia	Vital.	
Urgencia	Inmediatamente.	
Estado	Validado.	

Tabla 2.104: Requisito de información IRQ-1. Registros.

ID	**IRQ-2**	
Nombre	Consultas.	
Descripción	Cada una de las consultas, junto con el código fuente que las generó si está disponible, que los usuarios pueden ejecutar sobre los registros.	
Dependencias	FRQ-1, FRQ-2, FRQ-3, FRQ-4, FRQ-6, FRQ-7, FRQ-8.	
Campos	Log asociado, propietario, nombre, descripción, código fuente, restricciones y configuraciones asociadas.	
Tiempo de vida	**Medio** 15 días	**Máximo** ∞
Ocurrencias simultáneas	**Medio** 10	**Máximo** ∞
Importancia	Vital.	
Urgencia	Hay presión.	
Estado	Validado.	
Estabilidad	Alta.	

Tabla 2.106: Requisito de información IRQ-2. Consultas.

ID	IRQ-3	
Nombre	Restricciones.	
Descripción	Cada una de las restricciones, junto con el código fuente que las generó si está disponible, que se pueden aplicar sobre el algoritmo.	
Dependencias	FRQ-1, FRQ-2, FRQ-3, FRQ-4, FRQ-6, FRQ-7.	
Campos	Log y consulta asociados, propietario, nombre, descripción, lenguaje, código fuente y configuraciones asociadas.	
Tiempo de vida	**Medio** 15 días	**Máximo** ∞
Ocurrencias simultáneas	**Medio** 10	**Máximo** ∞
Importancia	Importante.	
Urgencia	Hay presión.	
Estado	Validado.	
Estabilidad	Alta.	

Tabla 2.108: Requisito de información IRQ-3. Restricciones.

ID	IRQ-4	
Nombre	Configuración.	
Descripción	Cada configuración que escoge el usuario para ejecutar el algoritmo. No resulta tan esencial para la aplicación porque podría guardarse temporalmente para la ejecución y no mostrarse posteriormente.	
Dependencias	FRQ-1, FRQ-3, FRQ-4, FRQ-9, FRQ-10.	
Campos	Log, consulta y restricción asociados, propietario, nombre, descripción, fitness mínimo, configuración de extensiones al algoritmo descritas posteriormente y resultados asociados.	
Tiempo de vida	**Medio** 15 días	**Máximo** ∞
Ocurrencias simultáneas	**Medio** 10	**Máximo** ∞
Importancia	Quedaría bien.	
Urgencia	Puede esperar.	
Estado	Validado.	
Estabilidad	Alta.	

Tabla 2.110: Requisito de información IRQ-4. Configuración.

ID	IRQ-5	
Nombre	Resultados.	
Descripción	Los resultados del algoritmo.	
Dependencias	FRQ-1, FRQ-2, FRQ-4, FRQ-9, FRQ-10, FRQ-11.	
Campos	Nº de trazas que cumplen, y de casos pudiendo contar varios en cada traza, fitness medio de las trazas válidas, los IDs de referencia a trazas válidas (opcional), y los resultados de extensiones al algoritmo (opcional).	
Tiempo de vida	**Medio** 15 días	**Máximo** ∞
Ocurrencias simultáneas	**Medio** 10	**Máximo** ∞
Importancia	Vital.	
Urgencia	Hay presión.	
Estado	Validado.	
Estabilidad	Alta.	

Tabla 2.112: Requisito de información IRQ-5. Resultados.

ID	IRQ-6	
Nombre	Tareas.	
Descripción	Las tareas se deben almacenar mientras estén en ejecución, con el nombre y el log asociado a la misma.	
Dependencias	FRQ-4, FRQ-15, FRQ-16.	
Campos	Nombre de tarea, registro completo actualizado (formado por un conjunto de porcentajes, mensajes, errores y campos específicos a la tarea).	
Tiempo de vida	**Medio** 2 horas	**Máximo** ∞
Ocurrencias simultáneas	**Medio** 2	**Máximo** ∞
Importancia	Quedaría bien.	
Urgencia	Puede esperar.	
Estado	Validado.	
Estabilidad	Alta.	

Tabla 2.114: Requisito de información IRQ-6. Tareas.

ID	IRQ-7	
Nombre	Usuarios.	
Descripción	Las usuarios de la aplicación y su información asociada, con su configuración global de la aplicación.	
Dependencias	FRQ-1, FRQ-3, FRQ-4, FRQ-12, FRQ-13, FRQ-14.	
Campos	Nombre, contraseña cifrada, email (opcional), rol, bloqueado, fecha de registro, configuración tiempo email.	
Tiempo de vida	**Medio** 1 mes	**Máximo** ∞
Ocurrencias simultáneas	**Medio** 2	**Máximo** ∞
Importancia	Importante.	
Urgencia	Inmediatamente.	
Estado	Validado.	
Estabilidad	Alta.	

Tabla 2.116: Requisito de información IRQ-7. Usuarios.

2.3.2. Requisitos funcionales

Un requisito funcional define una característica que un sistema debe ofrecer, definiendo el comportamiento interno del software: cálculos, detalles técnicos, manipulación de datos y otras funcionalidades específicas que muestran cómo los casos de uso descritos en la Sección 2.2: Casos de uso serán llevados a la práctica. Se especifican menos que los casos de uso para reducir el tamaño de la memoria y evitar duplicaciones. El significado del campo importancia puede necesitar aclaración:

Esencial: un requisito necesario para aceptar el producto y si no se cumpliese, el proyecto no se podría entregar.

Deseable: un requisito es deseable cuando se considera una mejora al producto pero ésta no es imprescindible para la aceptación del producto final.

Opcional: los requisitos que aumentan la calidad del sistema, pero se aceptará si no se consiguen implementar en los plazos existentes.

Posteriormente se mostrará la matriz de trazabilidad entre ambos apartados, muy útil para completar la información proporcionada inspeccionando los casos de uso.

ID	FRQ-1
Nombre	Listar registros, consultas, restricciones, configuraciones y resultados, usuarios y las actividades de registros.
Importacia	Esencial.

Continúa en la siguiente página

Continuado de la página anterior

Validación	Se dispone de la lista de recursos solicitada y al seleccionar un log, se dispone de la lista de actividades del registro solicitado.
ID	**FRQ-2**
Nombre	Importación y exportación de registros, consultas y restricciones, y exportación de resultados.
Importacia	Deseable.
Validación	Se puede ver que el elemento importado aparece en el listado al importar y que se completa la descarga correctamente al exportar.
ID	**FRQ-3**
Nombre	Modificar metadatos de registros, consultas, restricciones, configuraciones y usuarios.
Importacia	Deseable.
Validación	Se consiguen editar los metadatos, viendo los nuevos tras la edición.
ID	**FRQ-4**
Nombre	Borrar registros, consultas, restricciones, configuraciones y resultados, usuarios, cancelar tareas .
Importacia	Esencial.
Validación	Se ve que el elemento ya no se encuentra.
ID	**FRQ-5**
Nombre	Adaptar registros de otras clases Java (inVerbis, ProDiGen).
Importacia	Opcional.
Validación	El desarrollador dispone del modelo adaptado usable por el algoritmo.
ID	**FRQ-6**
Nombre	Crear consultas y restricciones a partir de sus respectivos lenguajes.
Importacia	Deseable.
Validación	Se pueden ver la consulta y la restricción creadas.
ID	**FRQ-7**
Nombre	Obtener código fuente de consultas y restricciones.
Importacia	Deseable.
Validación	Se puede ver la consulta con el código fuente.
ID	**FRQ-8**
Nombre	Previsualizar modelos de consultas.
Importacia	Deseable.
Validación	Se puede previsualizar la consulta como grafo y como árbol de análisis sintáctico editable.
ID	**FRQ-9**
Nombre	Configurar y ejecutar algoritmo.

Continúa en la siguiente página

Continuado de la página anterior

Importacia	Esencial.
Validación	Se ha iniciado la ejecución con esta configuración y se comprueba en los resultados guardados.
ID	**FRQ-10**
Nombre	Ver resultados y configuraciones.
Importacia	Esencial.
Validación	Se puede ver el resultado y otros datos asociados.
ID	**FRQ-11**
Nombre	Guardar resultados como log.
Importacia	Deseable.
Validación	Se ve el nuevo log en la lista, formado por las trazas que cumplen las condiciones del log anterior.
ID	**FRQ-12**
Nombre	Bloquear cuenta.
Importacia	Opcional.
Validación	Ya no se puede iniciar sesión (o puede).
ID	**FRQ-13**
Nombre	Cambiar rol.
Importacia	Opcional.
Validación	Se tiene el nuevo rol.
ID	**FRQ-14**
Nombre	Crear usuario o invitado.
Importacia	Esencial
Validación	Se puede iniciar sesión con la nueva cuenta.
ID	**FRQ-15**
Nombre	Consultar registro de una tarea en tiempo real.
Importacia	Esencial
Validación	Se ve el progreso y mensajes informativos de la tarea.
ID	**FRQ-16**
Nombre	Enviar email.
Importacia	Esencial
Validación	Se recibe el correo que indica el progreso y mensajes informativos de la tarea, además de un enlace a la interfaz.

Tabla 2.117: Requisitos funcionales.

2.3.3. Matriz de trazabilidad casos de uso/requisitos funcionales

Los datos son accesibles en la Tabla 2.119.

FRQ:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
CU-1	X															
CU-2	X															
CU-3		X														
CU-4		X														
CU-5		X														
CU-6			X													
CU-7				X												
CU-8					X											
CU-9						X										
CU-10							X									
CU-11								X								
CU-12	X															
CU-13		X														
CU-14		X														
CU-15		X														
CU-16			X													
CU-17				X												
CU-18						X										
CU-19							X									
CU-20	X															
CU-21		X														
CU-22		X														
CU-23		X														
CU-24			X													
CU-25				X												
CU-26										X						
CU-27	X															
CU-28				X												
CU-29									X							
CU-30									X							
CU-31									X							
CU-32	X															
CU-33										X						
CU-34		X														
CU-35			X													
CU-36				X												
CU-37											X					
CU-38												X				
CU-39													X			
CU-40	X															
CU-41														X		
CU-42														X		
CU-43			X													
CU-44				X												
CU-45															X	
CU-46																X
CU-47				X												

Tabla 2.119: Matriz de trazabilidad casos de uso\requisitos funcionales.

2.3.4. Requisitos no funcionales

Un requisito no funcional describe unos criterios que pueden usarse para validar la operación de un proyecto en lugar de sus comportamientos específicos, es decir, proponen unos límites de rendimiento, disponibilidad tiempos de respuesta, capacidad de almacenamiento, etc. Se detallan en la Tabla 2.120: Requisitos no funcionales.

ID	**NRQ-1**
Nombre	Almacenar con MongoDB [8] los datos.
Importacia	Esencial.
Validación	Se dispone de todos los requisitos de información en la base de datos MongoDB, excepto las tareas que no es necesario almacenarlas en base de datos.
ID	**NRQ-2**
Nombre	Usar al menos dagre [14] para la representación de grafos.
Importacia	Deseable.
Validación	Se pueden ver y previsualizar cuando se están editando los grafos del usuario usando la tecnología dagre. Se podría añadir una segunda implementación con dot [13].
ID	**NRQ-3**
Nombre	Implementación modular.
Importacia	Esencial.
Validación	Se pueden usar los módulos desarrollados en cada incremento por separado sin grandes dependencias entre los mismos, y que sean extensibles en la medida de lo posible.
ID	**NRQ-4**
Nombre	La interfaz se desarrolla en React [11].
Importacia	Esencial.
Validación	La interfaz está desarrollada usando React para renderizar sus componentes.
ID	**NRQ-5**
Nombre	La interfaz se desarrolla con Redux [9].
Importacia	Esencial.
Validación	La interfaz está desarrollada usando Redux para almacenar el estado para datos que se usan globalmente en la aplicación, mientras que para datos locales a los componentes se aprovecha el uso de React.
ID	**NRQ-6**
Nombre	La interfaz es usable.
Importacia	Deseable.

Continúa en la siguiente página

Continuado de la página anterior

Validación	La interfaz supera un cuestionario SUS con más de 8/10 puntos de media enviado al menos a 2 personas externas al proyecto con conocimientos sobre el funcionamiento de la herramienta.
ID	**NRQ-7**
Nombre	Uso de Spark en Java.
Importacia	Esencial.
Validación	El algoritmo debe usar Spark al menos para la paralelización del código base, y es deseable que también lo use para otras acciones como adaptaciones de registros.
ID	**NRQ-8**
Nombre	Multiplataforma.
Importacia	Deseable.
Validación	El proyecto debe ser completamente ejecutable en distintas plataformas, principalmente Linux y Windows.
ID	**NRQ-9**
Nombre	Desarrollo del servicio con Spring [19] y Spring Boot [20].
Importacia	Deseable.
Validación	El servidor debe haberse desarrollado usando estas tecnologías.
ID	**NRQ-10**
Nombre	Uso de ANTLR [18] para los lenguajes, y Kotlin [16] para el editor de lenguajes de la interfaz.
Importacia	Deseable.
Validación	El servidor debe haberse desarrollado usando estas tecnologías.

Tabla 2.120: Requisitos no funcionales.

Capítulo 3

Gestión del proyecto

En este capítulo se recoge toda la documentación relacionada con la gestión del proyecto. Primero se aclara el alcance de la herramienta, indicando las partes que no entran en el proyecto. En la siguiente sección se listan los diferentes riesgos identificados que pueden darse y su estudio. Después se incluye el documento de gestión de la configuración para este proyecto. A continuación, se hará la planificación temporal y de costes. Por último, se describe el plan de gestión de las comunicaciones que se aplicará a lo largo del proyecto.

3.1. Alcance

3.1.1. Descripción

Desde la versión 5 del PMBOK [17] se incluye en actividades iniciales la "Identificación de los implicados" como parte de la definición del alcance, y esta se puede visualizar en la Subsección 3.2.1.

El proyecto tiene un alcance bastante amplio, basado en los 3 objetivos principales. El alcance del servidor y la interfaz gráfica debe ser proporcionar al desarrollador y al usuario con las habilidades para utilizar el sistema de forma sencilla, por lo que me centro en el alcance del propio algoritmo y sus extensiones.

La herramienta debe permitir cargar registros y consultas en distintos formatos comunes y ejecutar el algoritmo sobre todos ellos. El algoritmo se debe poder ejecutar en local o desplegar en un clúster optimizado en el que cada traza del registro se procesa en paralelo en los distintos nodos. Los registros y consultas se deben poder exportar también en distintos formatos. Los resultados se pueden visualizar en la interfaz o exportar.

El usuario debe ser capaz de crear consultas de cero a partir de un lenguaje de declaración de consultas y un editor avanzado en la interfaz que realiza análisis sintáctico, puede mostrar un árbol para editar el lenguaje aplicando acciones sobre los nodos del mismo y permite la previsualización sin necesidad de realizar peticiones al servicio.

Además, el usuario debe ser capaz de introducir registros con *KPIs* e introducir restricciones en un lenguaje definido previamente que permitan un control mucho mayor asociando condiciones de KPIs a los eventos sobre los que se está ejecutando el algoritmo. Este lenguaje debe sufrir una compilación previa a la ejecución para evitar el parseado en el bucle más interno del algoritmo, lo cual tendría unas consecuencias de pérdida de rendimiento importante para el algoritmo. Para este lenguaje también se desarrollará un editor similar al anterior, sin la previsualización que no tiene sentido.

El objetivo principal de la interfaz gráfica es permitir un acceso eficiente, reactivo y con una buena experiencia de usuario a todas las características del algoritmo y el servidor descritas anteriormente. Además, se debe buscar que en la medida de lo posible sea accesible desde dispositivos móviles fácilmente (interfaz adaptativa).

Un ejemplo de interacción con el sistema que debería estar soportado podría ser en una tienda, en la que interesa saber en cuantos casos de los que se han guardado en sus registros, un usuario cualquiera compró determinado producto (guardado en una *KPI* asociada al evento de compra) y, lo devolvió o compró otro determinado producto, por el mismo usuario, y con un límite de tiempo de 3 días.

3.1.2. Criterios de aceptación

Estos se definen para cada requisito en el Capítulo 2, por lo que no se describen aquí.

3.1.3. Entregables

Al tratarse de un proyecto desarrollado por incrementos, los entregables son la suma de los entregables de cada incremento, que se pueden ver en la Tabla 3.2.

3.1.4. Exclusiones del proyecto

No se aplican exclusiones en el proyecto buscando maximizar el valor del proyecto.

3.1.5. Restricciones

Las restricciones principales se pueden ver en la Tabla 3.4.

3.2. Plan de gestión de las comunicaciones

Gestionar las comunicaciones para este proyecto, aunque haya pocos interesados, puede mejorar la calidad del mismo, reduciendo las probabilidad de fracaso del mismo. Este se crea antes de la gestión de riesgos ya que los interesados pueden generar riesgos, como se verá en la siguiente sección.

Nombre	Descripción
Algoritmo	La implementación del algoritmo base en un formato en el que pueda ser usado como librería por otros programas sin necesidad de depender del servidor.
Servidor	El conjunto de servicios que permiten interactuar con el algoritmo a través de peticiones HTTP. Este podrá ser usado para automatizar tareas por desarrolladores.
Interfaz gráfica	La interfaz de usario desarrollada con tecnologías web que realiza peticiones sobre el servidor.
Memoria	Documento que detalla todo el trabajo realizado en el proyecto.
Documentación	Consiste en la descripción de las interfaces orientadas para facilitar el uso por desarrolladores de algoritmo y servidor.
Manual de instalación	Manual que permite a un usuario instalar los componentes del proyecto en sus dispositivos.
Manual de usuario	Manual de uso para usuarios centrado en la interfaz gráfica. Podría entregarse como parte de la propia interfaz de forma que la ayuda esté integrada en ella.

Tabla 3.2: Entregables.

Nombre	Descripción
Disponibilidad	El proyecto debe compaginarse con el resto del curso, es decir, la asignatura de aspectos legales y las prácticas en empresa (de 5 horas diarias en días laborables de abril al 7 de junio). Además también existe una beca de colaboración en el CiTIUS de 3 horas diarias de dedicación durante todo el proyecto.
Plazos	El proyecto se entregará como máximo el 28 de junio.
Mínimos	Se harán un mínimo de 401,25 horas de trabajo.

Tabla 3.4: Restricciones.

3.2.1. Gestión de interesados

Este es un apartado clave para poder construir la gestión de las comunicaciones sobre estos datos. Como indica el PMBOK [17] estudiando en gestión de proyectos, los interesados son:

> Las personas y organizaciones que participan de forma activa en el proyecto o cuyos intereses pueden verse afectados como resultado de la ejecución del proyecto o de su conclusión.

En este apartado se resumen los resultados de todos los procesos de gestión de los interesados: Identificar a los Interesados, Planificar el Involucramiento de los Interesados, Gestionar el Involucramiento de los Interesados y Monitorear el Involucramiento de los Interesados[1].

Para ello se genera una matriz que muestra toda la información recopilada al identificar a los interesados. En este caso no existen muchos interesados ni varían a lo largo del proyecto (por lo que el tercer y cuarto procesos no destacan), aunque normalmente variarían y la gestión de interesados es un proceso continuo. La matriz, que incluye los resultados de los procesos de identificación (primer grupo de columnas) y gestionar el involucramiento de los interesados (segundo y tercer grupo de columnas), se encuentra en la Tabla 3.6.

3.2.2. Planificación de la información y comunicaciones

Este proceso permite un flujo de información eficaz y eficiente entre el equipo del proyecto y los interesados ya recogidos en la Subsección 3.2.1. Para ello, se deben definir claramente conceptos como quién necesita la información, qué información se distribuye, cuándo se proporciona, etc.

Existen tres métodos de comunicación que se pueden dar en un proyecto: interactiva, que realiza un intercambio de información multidireccional entre dos o más partes; *push*, que se envía a receptores específicos; y *pull*, que se pone a disposición de los receptores y estos acceden a ella cuando la necesiten.

Los principales canales de comunicación de este proyecto son interactivos. Tanto los tutores como el desarrollador podrán iniciar la comunicación de forma interactiva. La comunicación con los miembros de inVerbis que buscar integrar este proyecto como biblioteca es de tipo *pull* porque la última versión del proyecto se pone a su disposición a través de un repositorio nexus ya explicado, al que pueden acceder cuando deseen para actualizar la dependencia. También se les da acceso a documentación y se responden dudas cuando lo solicitan. La comunicación con los futuros usuarios que es de tipo *pull* por motivos similares. La Tabla 3.8 indica el resto de características sobre las comunicaciones.

[1] *La guía de los fundamentos para la dirección de proyectos (Guía del PMBOK)*, p. 503 [17]

Identificación y contacto					Requisitos e intereses				Clasificación	
Nombre	**Empresa/ grupo**	**Rol**	**Localización**	**Contacto**	**Requerimientos**	**Expectativas principales**	**Influencia potencial**	**Fase de mayor interés**	**Interno/ externo**	**Apoyo/ neutral/ opositor**
Jacobo Casas Ramos	———	Autor	EmprendeLab CiTIUS	jacobo .casas .ramos @rai.usc .es	Compromiso con el proyecto	Completar el proyecto en el plazo indicado	Alta	Todas	Interno	Apoyo
Manuel Lama Penin y Manuel Mucientes Molina	CiTIUS	Tutor/ Cliente	Despachos 106 / 110 CiTIUS	{manuel .lama, manuel .mucientes} @usc.es	Tutorización	Finalizar exitosamente el proyecto	Alta	Todas	Interno	Apoyo
Álvaro Porto Ares, Andrea Cascallar Fuentes...	inVerbis	Desarrollador	EmprendeLab CiTIUS	{alvaro. porto. ares, andrea. cascallar. fuentes...} @usc.es	———	Disponer del proyecto para su uso como biblioteca	Alta	Finalización del proyecto	Externo	Apoyo
Futuros usuarios	Usuarios	Usuario	———	———	———	Disponer del proyecto	Baja	Finalización del proyecto	Externo	Neutral

Tabla 3.6: Matriz de interesados.

Nombre	Propósito de la comunicación	Contenido	Nivel de detalle	Importancia	Periodicidad	Emisor	Receptor	Formato	Idioma	Métodos/ tecnologías
Jacobo Casas Ramos	Registrar el progreso del proyecto	Detalles de implementación del proyecto	El máximo posible	Alta	Continuamente	Autor	Autor	Digital	Español	Control de cambios, IDE, cronograma...
Manuel Lama Penin y Manuel Mucientes Molina	Informar sobre el progreso y buscar mejoras al proyecto	Demostraciones e informes sobre el resultado	Medio	Muy alta	Cada 1-2 semanas, de forma irregular	Autor	Tutores	Digital y en persona	Español	Correo electrónico y Reuniones
Álvaro Porto Ares, Andrea Cascallar Fuentes...	inVerbis	Cliente	Medio	Alta	Irregular, hacia el final del proyecto	Autor	Usuarios	Documentos electrónicos e impresos	Español	Repositorio Nexus, manuales y reuniones...
Futuros usuarios	Usuarios	Cliente	Medio	Alta	Irregular, hacia el final del proyecto	Jefe del proyecto	Usuarios	Documentos electrónicos e impresos	Español	Manuales de instalación o uso, tutoriales...

Tabla 3.8: Matriz de comunicaciones.

3.2.3. Reuniones

Además de la frecuencia y métodos de comunicación, es necesario indicar la duración de las reuniones, ya que reuniones con una mala organización se pueden alargar reduciendo la productividad del trabajo.

Las reuniones se realizan generalmente en el despacho de Manuel Lama Penín del CiTIUS, ya que los integrantes de la misma trabajan en el edificio. Las reuniones son de aproximadamente 1 hora dependiendo de las actualizaciones al proyecto que se vayan a mostrar.

Se realizan aproximadamente cada 1-2 semanas, de forma irregular. En las etapas iniciales de cada incremento (etapas de análisis) se espera una mayor frecuencia de reuniones para aclarar los conceptos y el resultado, mientras que durante el resto de etapas la interacción será menor debido a que se escogió un ciclo de vida por incrementos.

3.3. Gestión de riesgos

> Un riesgo de un proyecto es un evento o condición inciertos que, si se produce, tiene un efecto positivo o negativo sobre al menos un objetivo del proyecto, como tiempo, coste, alcance o calidad. [17]

Un efecto es la materialización de una amenaza por la que sufren un impacto alguno de los activos del proyecto. Por ello, se comenzará listando los activos y amenazas del proyecto, que facilitan la identificación de riesgos relacionados con estos. Después se analizarán cualitativamente los riesgos, se escogerán los indicadores y las estrategias de control de los riesgos.

3.3.1. Activos

A1. Autor.

A2. Tutores.

A3. Portátil autor.

A4. GitLab.

A5. Nexus.

A6. Clúster Big Data.

A7. Otras instalaciones CiTIUS.

A8. Relaciones con inVerbis.

A9. Conexión a la instalación eléctrica.

A10. Conexión a la Internet.

A11. Producto final.

3.3.2. Amenazas

M1. Caída de luz.
M2. Accidentes.
M3. Caída servidor GitLab.
M4. Caída repositorio Nexus.
M5. Caída clúster CiTIUS.
M6. Ataque informático.
M7. Fallo conexión a Internet.
M8. Problema tecnología elegida.
M9. Incumplir plazo.
M10. Falta de experiencia.
M11. Cancelación proyecto.
M12. Pérdida de datos.
M13. Falta de compromiso.
M14. Cambio de requisitos.
M15. Dificultad de representación de grafos.
M16. Complejidad de los lenguajes a desarrollar.
M17. Fallos de comunicación.

3.3.3. Identificación de riesgos

No se incluyen riesgos relacionados con acceso a internet, corriente eléctrica y similares por su baja probabilidad e impacto.

Activos	Amenazas	Riesgo	Espec.[2]
A1,A2,A3	M2	R-1. Incapacidad de implicado o material por accidentedente.	Enlace
A1	M10	R-2. El autor no tiene experiencia en el proyecto, retrasando los resultados.	Enlace
A1,A2	M8,M11,M13	R-3. Se cancela el proyecto.	Enlace
A1,A2	M13	R-4. Retrasos en el proyecto por falta de compromiso.	Enlace
A1,A2,A11	M9,M10	R-5. Planificación demasiado optimista.	Enlace

Continúa en la siguiente página

Continuado de la página anterior

Activos	Amenazas	Riesgo	Espec.
A3	M6	R-6. Malware en el ordenador retrasa o impide continuar el proyecto.	Enlace
A3,A4,A5,A6,A7	M8	R-7. Incompatibilidades impiden el progreso del proyecto.	Enlace
A3,A4,A5,A6,A7	M12	R-8. Pérdida de datos causa retrasos.	Enlace
A4	M1,M3,M7	R-9. El servidor de GitLab deja de estar disponible.	Enlace
A5	M1,M4,M7	R-10. El repositorio Nexus deja de estar disponible.	Enlace
A6	M1,M5,M7	R-11. El cluster del CiTIUS deja de estar disponible.	Enlace
A7	M1,M6,M7	R-12. Pérdida de servicios del CiTIUS (nextcloud, correo...).	Enlace
A8,A11	M9	R-13. Retrasarse demasiado en la entrega a inVerbis.	Enlace
A11	M9	R-14. No poder entregar y presentar el proyecto en la fecha prevista	Enlace
A11	M14	R-15. Retrasos por cambio de requisitos.	Enlace
A11	M15	R-16. No conseguir representar grafos en la interfaz.	Enlace
A11	M16	R-17. No conseguir desarrollar el analizador sintáctico o el editor en la interfaz.	Enlace
A1,A2,A11	M17	R-18. La falta de o mala comunicación causan malentendidos y retrasos en el proyecto.	Enlace

Tabla 3.9: Identificación de riesgos.

3.3.4. Análisis cualitativo

Los riesgos se clasifican en tres categorías según su probabilidad:

Alto. Más del 70 % de probabilidad de aparición.

Medio. Entre el 30 % y el 70 % de probabilidad.

Bajo. Menos del 30 % de probabilidad.

También se clasifican en 3 categoría según su nivel de impacto:

Grave. La aparición del riesgo puede suponer la cancelación o retraso del proyecto de forma que no se entregaría en los plazos previstos.

Moderado. La aparición del riesgo afectaría a la planificación sustancialmente, retrasando más de 2 días la fecha esperada de finalización.

Leve. La aparición del riesgo crearía retrasos que no afectan al proyecto tanto como el nivel moderado.

Por último se indica el nivel de exposición del riesgo, que depende de los dos anteriores:

Prob\Impacto	Grave	Moderado	Leve
Alta	Alta	Alta	Media
Media	Alta	Media	Baja
Baja	Media	Baja	Baja

Tabla 3.11: Exposición al riesgo.

3.3.5. Estrategias

Las acciones realizables sobre un riesgo para tratarlo se clasifican en:

Evitar. es eliminar la amenaza completamente realizando cambios sobre el proyecto.

Mitigar. implica reducir la probabilidad o el impacto que un riesgo puede tener sobre los activos a los que afecta realizando o planificando acciones.

Transferir. es una estrategia común en la gestión de proyectos, en casos en los que no se puede cambiar el plan de proyecto y no se sabe cómo reducir la probabilidad del riesgo, la solución puede ser aceptarlo y asumir que puede ocurrir. Esta aceptación puede ser pasiva (no hacer nada) o activa (reservar un fondo para contingencias que se utilizará si el riesgo aparece)..

[2]Enlace a la especificación.

Aceptar. consiste en derivar la responsabilidad del mismo a un tercero. Esto no elimina el riesgo, sino que traslada a otra entidad el trabajo de tratarlo..

Se tratará de identificar más una estrategia para cada riesgo siempre que sea posible.

3.3.6. Resultados de análisis y especificación de riesgos

En este apartado se realiza el análisis explicado, indicando también el indicador que se seguirá para monitorizarlo y las estrategias a seguir en caso de que se materialice el riesgo:

ID	**R-1** **Identificación**
Nombre	Incapacidad de implicado o material por accidente.
Descripción	Se producen bajas por un accidente o se pierde material en un accidente de forma que no se puede continuar el proyecto.
Probabilidad	Baja
Impacto	Alto
Exposición	Media
Indicador	El programador no puede realizar las tareas durante más de tres días.
Mitigar	No realizar grandes actualizaciones ni cambios en el sistema de desarrollo durante el proyecto. Esto reduce la probabilidad de ocurrencia.
Aceptar	Detención del proyecto temporalmente y replanificación cuando se pueda continuar.

Tabla 3.13: Especificación de R-1. Imposibilidad de continuar el proyecto por accidente.

ID	R-2	Identificación
Nombre	El autor no tiene experiencia en el proyecto, retrasando los resultados.	
Descripción	Ocurren retrasos no planificados por la necesidad de formación en el proyecto en muchas tecnologías: Spark, Kotlin, Mongo...	
Probabilidad	Media	
Impacto	Medio	
Exposición	Media	
Indicador	El programador no puede realizar las tareas durante más de dos días.	
Evitar	La formación se realiza al comienzo del proyecto, minimizando el impacto, y la planificación tiene en cuenta posibles retrasos. El impacto es ahora bajo.	
Aceptar	Detención del proyecto temporalmente y replanificación cuando se pueda continuar.	

Tabla 3.15: Especificación de R-2. El autor no tiene experiencia en el proyecto, retrasando los resultados.

ID	R-3	Identificación
Nombre	Se cancela el proyecto.	
Descripción	Se decide no continuar el proyecto por los tutores, el autor o simplemente falta de compromiso con el mismo.	
Probabilidad	Baja	
Impacto	Alto	
Exposición	Media	
Indicador	Se decide cancelar el proyecto.	
Mitigar	Realizar un estudio de viabilidad inicial. Esto reduce la probabilidad de ocurrencia del riesgo.	

Tabla 3.17: Especificación de R-3. Se cancela el proyecto.

ID	R-4	Identificación
Nombre	Retrasos en el proyecto por falta de compromiso.	
Descripción	Ocurren retrasos no planificados por no cumplir con los horarios programados semanalmente debido a una falta de dedicación al proyecto.	
Probabilidad	Baja	
Impacto	Medio	
Exposición	Baja	
Indicador	El programador no realiza tareas durante más de dos días o trabaja menos de 15h en una semana (30h programadas por semana).	
Mitigar	Volver a enfocar el proyecto para aumentar el interés cuando se detecte el indicador. Esto reduce la probabilidad.	

Tabla 3.19: Especificación de R-4. Retrasos en el proyecto por falta de compromiso.

ID	R-5	Identificación
Nombre	Planificación demasiado optimista.	
Descripción	En el Gantt se ha asignado un tiempo más corto de lo que se necesita para realizar las tareas, por lo que se retrasa todo el proyecto, posiblemente retrasando la entrega	
Probabilidad	Media	
Impacto	Alto	
Exposición	Alta	
Indicador	El programador lleva un retraso de más de cinco días sobre la línea base planteada.	
Evitar	Se reducen las funcionalidades opcionales siempre que se cumplan los objetivos indicados. Este cambio de planificación implicaría reducir la probabilidad a baja.	
Mitigar	Se deja un colchón de tiempo para permitir ligeros retrasos. El impacto se reduce a medio.	
Aceptar	Se entregará el proyecto en el siguiente plazo si ocurre esto.	

Tabla 3.21: Especificación de R-5. Planificación demasiado optimista.

ID	R-6	Identificación
Nombre	Malware en el ordenador retrasa o impide continuar el proyecto.	
Descripción	Se instala malware que borra los datos o bloquea el dispositivo de forma que no se puede continuar trabajando.	
Probabilidad	Media	
Impacto	Alto	
Exposición	Alta	
Indicador	El programador lleva un retraso de más de cinco días sobre la línea base planteada.	
Evitar	Se usa una máquina virtual para manejar ciertos documentos (con copia en el repositorio) que deben tratarse en Windows (como el Gantt o algún diseño), de la que se hacen *snapshots* cada vez que se progresa significativamente. Se reduce la probabilidad para ciertos documentos.	
Mitigar	No realizar grandes actualizaciones ni cambios en el sistema de desarrollo durante el proyecto. Además se realizarán subidas del código continuas (al menos diarias) al repositorio remoto. La probabilidad se reduce a baja.	
Aceptar	Se reinstalará el sistema base y se recuperarán los datos del repositorio remote, que continene todos los datos del proyecto.	

Tabla 3.23: Especificación de R-6. Malware en el ordenador retrasa o impide continuar el proyecto.

ID	R-7	Identificación
Nombre	Incompatibilidades impiden el progreso en el proyecto.	
Descripción	No se puede seguir trabajando o hay que buscar una alternativa porque tecnologías como Spark, ANTLR, otras bibliotecas... o dispositivos y servicios como el ordenador del desarrollador, el clúster del CiTIUS, Nextcloud, GitLab, Nexus... no son compatibles entre sí.	
Probabilidad	Alta	
Impacto	Alto	
Exposición	Alta	
Indicador	Se descubren problemas de compatibilidad en las tecnologías durante la formación o implementación.	
Mitigar	Durante la formación se estudiará la compatibilidad. El impacto baja a medio, ya que se detectaría antes.	
Aceptar	Se buscará una alternativa a la herramienta o tecnología.	

Tabla 3.25: Especificación de R-7. Incompatibilidades impiden el progreso en el proyecto.

ID	R-8	Identificación
Nombre	Pérdida de datos causa retrasos.	
Descripción	El ordenador del desarrollador, el clúster del CiTIUS, Nextcloud, GitLab, Nexus... pierden los datos por error, lo cual fuerza a una reconstrucción de una parte del proyecto.	
Probabilidad	Media	
Impacto	Alto	
Exposición	Alta	
Indicador	Se recibe una notificación de futura pérdida de datos por mantenimiento u otros motivos o se pierden en el dispositivo propio.	
Evitar	Todos los datos están disponibles en el repositorio, el cual se copia al GitLab, por lo que tendrían que fallar el ordenador del autor y el GitLab en un periodo corto de tiempo para producir retrasos. La probabilidad se reduce a baja.	
Aceptar	Se buscará una alternativa a la herramienta o tecnología.	

Tabla 3.27: Especificación de R-8. Pérdida de datos causa retrasos.

ID	R-9	Identificación
Nombre	El servidor GitLab deja de estar disponible.	
Descripción	El servidor GitLab no es accesible durante un periodo importante de tiempo.	
Probabilidad	Baja	
Impacto	Alto	
Exposición	Media	
Indicador	Se recibe una notificación por mantenimiento u otros motivos.	
Aceptar	El repositorio se subira a un repositorio privado de la instancia de GitLab pública, lo cual instalará automaticamente el CI[3] y reducirá el impacto a bajo.	

Tabla 3.29: Especificación de R-9. El servidor GitLab deja de estar disponible.

ID	R-10	Identificación
Nombre	El repositorio Nexus deja de estar disponible.	
Descripción	El repositorio Nexus no es accesible durante un periodo importante de tiempo.	
Probabilidad	Baja	
Impacto	Alto	
Exposición	Media	
Indicador	Se recibe una notificación por mantenimiento u otros motivos.	
Evitar	La integración continua se construye de forma resistente a caídas del repositorio, usando las dependencias de la carpeta local que se mantienen actualizadas.	

Tabla 3.31: Especificación de R-10. El repositorio Nexus deja de estar disponible.

ID	R-11	Identificación
Nombre	El cluster del CiTIUS deja de estar disponible.	
Descripción	El cluster del CiTIUS no es accesible durante un periodo importante de tiempo.	
Probabilidad	Baja	
Impacto	Medio	
Exposición	Baja	
Indicador	Se recibe una notificación por mantenimiento u otros motivos.	
Aceptar	Se retrasa esta parte del proyecto, planificando otras tareas para seguir trabajando.	

Tabla 3.33: Especificación de R-11. El cluster del CiTIUS deja de estar disponible.

ID	R-12	Identificación
Nombre	Pérdida de servicios del CiTIUS (nextcloud, correo).	
Descripción	Otros servicios del CiTIUS no son accesibles durante un periodo importante de tiempo.	
Probabilidad	Baja	
Impacto	Medio	
Exposición	Baja	
Indicador	Se recibe una notificación por mantenimiento u otros motivos.	
Mitigar	Se realizarán comunicaciones en persona o a correos alternativos y se usará Google Drive para almacenamiento, reduciendo el impacto a bajo.	

Tabla 3.35: Especificación de R-12. Pérdida de servicios del CiTIUS (nextcloud, correo).

ID	R-13	Identificación
Nombre	Retrasarse demasiado en la entrega a inVerbis.	
Descripción	No cumplir los límites de plazo de entrega del proyecto como librería (incremento 1) a inVerbis o realizar una entrega parcial.	
Probabilidad	Baja	
Impacto	Bajo	
Exposición	Baja	
Indicador	Se solicita a inVerbis un plazo y quedan menos de 5 días para ese plazo sin haber terminado el incremento.	
Aceptar	Entregar el producto parcial e indicar lo que falta y cuando está planificado acabarlo.	

Tabla 3.37: Especificación de R-13. Retrasarse demasiado en la entrega a inVerbis.

ID	R-14	Identificación
Nombre	No poder entregar y presentar el proyecto en la fecha prevista.	
Descripción	Los retrasos o una planificación incorrecta llevan a incumplir el plazo de entrega.	
Probabilidad	Alta	
Impacto	Alto	
Exposición	Alta	
Indicador	La planificación va con suficiente retraso como para incumplir los plazos.	
Mitigar	Se puede reducir el alcance de funcionalidades que no afecten a los objetivos, como poder importar o exportar menos formatos y ser compatibles con menos proyectos externos, bajando el impacto a medio.	
Aceptar	Se entrega el proyecto en el siguiente plazo.	

Tabla 3.39: Especificación de R-14. No poder entregar y presentar el proyecto en la fecha prevista.

ID	R-15	Identificación
Nombre	Retrasos por cambio de requisitos.	
Descripción	Los tutores cambian los requisitos después de la etapa de análisis de los incrementos, de forma que es necesario volver a realizar etapas anteriores, provocando retrasos en el proyecto.	
Probabilidad	Alta	
Impacto	Alto	
Exposición	Alta	
Indicador	Se cambia un requisito.	
Mitigar	Se les recordará a los tutores que los requisitos no deben cambiar a lo largo de cada incremento, y se realizarán varias reuniones al inicio del incremento para dejar todos los requisitos.	
Aceptar	Se vuelve a planificar el proyecto con los cambios adecuados para los nuevos requisitos.	

Tabla 3.41: Especificación de R-15. Retrasos por cambio de requisitos.

ID	R-16	Identificación
Nombre	No conseguir representar grafos en la interfaz.	
Descripción	La falta de experiencia del autor de trabajo con grafos podría complicar en gran medida la representación de los grafos en la interfaz si la curva de aprendizaje de las tecnologías a usar es elevada.	
Probabilidad	Alta	
Impacto	Medio	
Exposición	Alta	
Indicador	En la formación se ve que las librerías son complejas de usar.	
Evitar	Se buscarán alternativas en la etapa de formación.	
Mitigar	Se intentará realizar una implementación manual de la representación de un grafo.	
Aceptar	Se imprimirán textualmente los grafos (indicando los nodos y los arcos entre ellos).	

Tabla 3.43: Especificación de R-16. No conseguir representar grafos en la interfaz.

ID	R-17	Identificación
Nombre	No conseguir desarrollar el analizador sintáctico o el editor en la interfaz.	
Descripción	Desarrollar un analizador sintáctico es una tarea compleja, y se usarán tecnologías nuevas para ello (Kotlin compilado a JavaScript), por lo que aumenta aún más la dificultad.	
Probabilidad	Media	
Impacto	Medio	
Exposición	Media	
Indicador	El estudio realizado en la etapa de formación no encuentra una forma viable de implementar este módulo del sistema.	
Aceptar	No se implementan esté modulo o las partes afectadas (editor de texto con colores, árbol sintáctico con acciones o previsualización) sino que se ofrece un área de texto en lugar de un editor avanzado.	

Tabla 3.45: Especificación de R-17. No conseguir desarrollar el analizador sintáctico o el editor en la interfaz.

ID	R-18	Identificación
Nombre	La falta de o mala comunicación causan malentendidos y retrasos en el proyecto.	
Descripción	Un mal análisis por una mala comunicación llevara a la necesidad de reiniciar un incremento.	
Probabilidad	Baja	
Impacto	Alto	
Exposición	Media	
Indicador	Se detecta un fallo de comunicación.	
Mitigar	Se les recordará a los tutores que los requisitos no deben cambiar a lo largo de cada incremento, y se realizarán varias reuniones al inicio del incremento para que quede muy claro el alcance del incremento y como se resolverá.	
Aceptar	Se planifican los cambios en el proyecto retrasando la finalización del mismo.	

Tabla 3.47: Especificación de R-18. La falta de o mala comunicación causan malentendidos y retrasos en el proyecto.

3.4. Gestión de la configuración

El propósito de este Plan de Gestión de la Configuración es proporcionar una descripción general de la organización, las actividades, las tareas generales y los objetivos de la Gestión de Configuración.

Aborda en control de cambios, la identificación de elementos de configuración y se proporcionan detalles sobre las actividades y los roles de la Gestión de la configuración, entre otros.

Este plan de gestión de la configuración está diseñado para proyectos de pequeño-mediano tamaño en los que trabaja un grupo reducido de unas 1-10 personas. Aunque el número de integrantes de un equipo de desarrollo no sea muy grande es importante definir un plan para manejar la gestión de la configuración correctamente.

3.4.1. Roles

Estos roles son llevados a cabo por integrantes del proyecto de forma organizada para mejorar la productividad y la calidad final.

Responsabilidades y asignaciones

Se definen los principales roles para las actividades de gestión de la configuración y las responsabilidades de cada uno de los roles, junto con las personas asignadas aparecen en la Tabla 3.49

3.4.2. Herramientas y descripción del soporte de las herramientas a la gestión de la configuración

Git

Todo el proyecto (código, memoria, documentos de diseño, planificación, figuras, etc.) se mantiene en un repositorio git, facilitando el manejo del mismo y reduciendo olvidos de subidas de código al servidor.

Al mantener todo el código en el mismo repositorio, se consigue evitar la existencia de versiones incompatibles entre los distintos proyectos o incrementos que forman el proyecto. Las versiones anteriores tendrán menos funcionalidades, pero sus proyectos interactuarán correctamente entre ellos, evitando la necesidad de crear y mantener una matriz de compatibilidad entre las distintas versiones de los distintos repositorios.

El uso de git como herramienta de control de versiones permite que no sean necesario cubrir plantillas de cambios ya que estas ya se indican automáticamente por git: autor, nombre del cambio, descripción de los cambios realizados, rama sobre la que se aplica, horas de trabajo aproximadas...

Rol	Responsabilidades	Asignado a
Gestor de Cambio	▪ Evaluar los cambios propuestos ▪ Gestionar los cambios aceptados ▪ Manejar toda la información relativa a los cambios sobre el proyecto	Jacobo Casas Ramos
Gestor de configuración	▪ Gestionar la planificación, control y seguimiento de todos los elemento de gestión de la configuración. ▪ Liderar el proceso de gestión de la configuración. ▪ Identificar elementos de la configuración.	Jacobo Casas Ramos
Dueño del repositorio	▪ Organizar los documentos del proyecto ▪ Mantener las versiones finales de los documentos ▪ Preparar los documentos para la entrega	Jacobo Casas Ramos

Tabla 3.49: Roles, responsabilidades y asignaciones de gestión de la configuración.

Cada cambio significativo realizado lleva un mensaje que explica que ha cambiado de forma clara y concisa.

GitLab

No es necesaria la existencia de un servidor para trabajar con git, pero la replicación en un servicio externo aumenta la seguridad de que se mantendrán todos los archivos seguros e incluso el historial de cambios de los mismos. Esto reduce el riesgo de pérdida de datos indicado anteriormente.

Se creó un repositorio remoto privado en el GitLab mantenido del CiTIUS para conservar la copia del repositorio local del proyecto por seguridad. Como los cambios son realizados por un solo desarrollador, se utiliza únicamente para la rama principal de los repositorios, salvo para algunas tareas de incorporación de funcionalidades que no se sabe si acabarán en el proyecto, en cuyo caso se crea una rama.

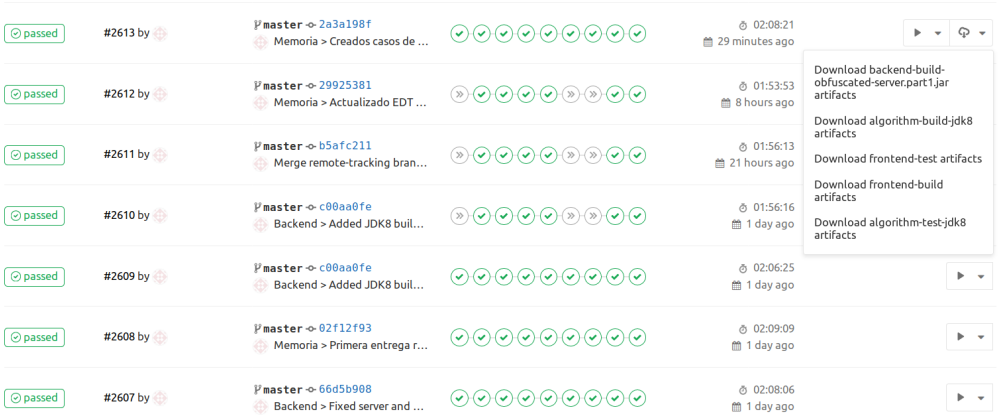


Figura 3.1: GitLab CI: 7 *pipelines* y la descargas de artefactos.

GitLab CI

Pero no sólo se usa de almacenamiento, sino que se desarrolla integración y despliegue continuo para todos los incrementos usando el sistema de integración continua incluido en GitLab. Cada vez que se realizan cambios en el proyecto, se ejecutan las pruebas de forma que se compruebe que no se han perdido funcionalidades en los cambios realizados. El sistema está optimizado usando una cache configurada en las rutas de dependencias que evita descargarlas en cada subida de código y en cada subtarea.

Además, la integración continua genera artefactos que deben ser manejados en la gestión de la configuración: *jars* con código compilado, *jars* con código optimizado y obfuscado, ficheros de interfaz *html*, *woff*, *js*... optimizado para producción, informes de pruebas realizadas y cobertura, etc. Todos estos se guardan de forma temporal (disponibles para su descarga durante 1 día desde la generación), y se mantienen organizados por GitLab, como se puede ver en la Figura 3.1.

Por último, el GitLab CI también se encarga de desplegar los ficheros JAR al repositorio nexus. En concreto, todas las tareas que realiza en cada subida de código son 36 en total, algunas de ellas manuales y 20 de ellas son para la subida de artefactos que superan los 100 MB, porque los JAR generados son demasiado grandes. Un *pipeline* normal tarda 2 horas debido a que se ejecutan las pruebas con información de debug para tomar medidas de cobertura más correctas. Las tareas detalladas se pueden ver en la Figura 3.2 y el código que las genera es .gitlab-ci.yml en la carpeta raíz del proyecto.

Maven/Gradle/yarn

Todas son herramientas de automatización de compilaciones de código, manejando dependencias del proyecto y configurando el entorno para que resulte muy sencillo desarrollar en distintos entornos. Al manejar las dependencias automáticamente están reduciendo en gran medida el trabajo de gestión de los documentos. El primer incremento se desarrolló con Maven, el segundo con

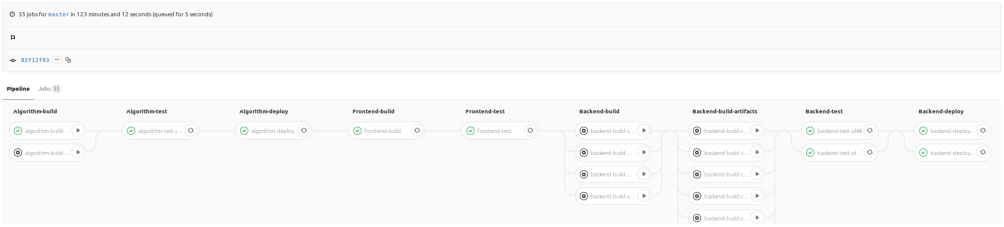


Figura 3.2: GitLab CI: las tareas de un *pipeline*.

Gradle y el tercero con yarn

Repositorio Nexus

Este es un repositorio que se mantiene en un servidor del CiTIUS de forma privada, lo cual permite compartir bibliotecas localmente de forma sencilla. Este se integra con Maven y Gradle (entre otros) para proporcionar las bibliotecas sin necesidad de compartirlas de forma pública.

En este proyecto se usa para actualizar algunas bibliotecas desarrolladas en el CiTIUS (*ProDiGen*, *WoMine*, *DataSources*, *DataWriters*, *GenericModel*, *Utils* y sus dependencias), de las cuales depende el proyecto. El algoritmo y el servidor también se despliegan automáticamente en este repositorio, facilitando el uso de los mismos por otros desarrolladores.

Infer

Infer es un analizador estático de código *Java*/*C++*/*Objective C*. Su trabajo es comprobar posibles errores en el código antes de ejecutarlo, como pueden ser las excepciones de puntero nulo, las fugas de recursos, el alcance de la anotación, los guardias de bloqueo que faltan y las condiciones de carrera de concurrencia en los códigos de Java.

Se usa para garantizar cierto nivel de calidad en el algoritmo (primer incremento completo) desarrollado para este proyecto. El informe está disponible en /conformancechecking/infer-out.

LaTeX

Es un lenguaje de composición de textos de alta calidad, estándar de facto para la creación de documentos y publicaciones de carácter científico.

TeXstudio

La memoria se desarrolla en LaTeX, y manejar esta cantidad de documentos sería muy difícil sin un IDE. TeXstudio es una herramienta gratuita que permite editar documentos LaTeX con características como múltiples cursores, autocompletado, marcadores, previsualización de enlace, asistentes, formateado de tablas, estructura de ficheros...

3.4.3. Nomenclatura

El uso de git como herramienta de control de versiones permite que no sean necesarios números de versiones en los ficheros ni registro de cambios en el propio fichero editado ya que todo esto lo maneja de una forma mucho mejor el propio sistema permitiendo recuperar los cambios realizados a cualquier fichero entre un par de fechas.

Solo podría resultar necesaria una nomenclatura especial cuando estos documentos se extraen del proyecto, como puede ser en el caso de la entrega. Para estos casos se define la nomenclatura siguiente:

$$nombre\text{-}documento\_aaMMdd\_\mathrm{v}version.extension$$

Donde *nombre-documento* tiene una función identificativa del mismo, *aaMMdd* es la fecha de modificación del documento, *version* coincide con la última tag de Git y *extension* es la que tendría el documento.

3.4.4. Estructura del proyecto

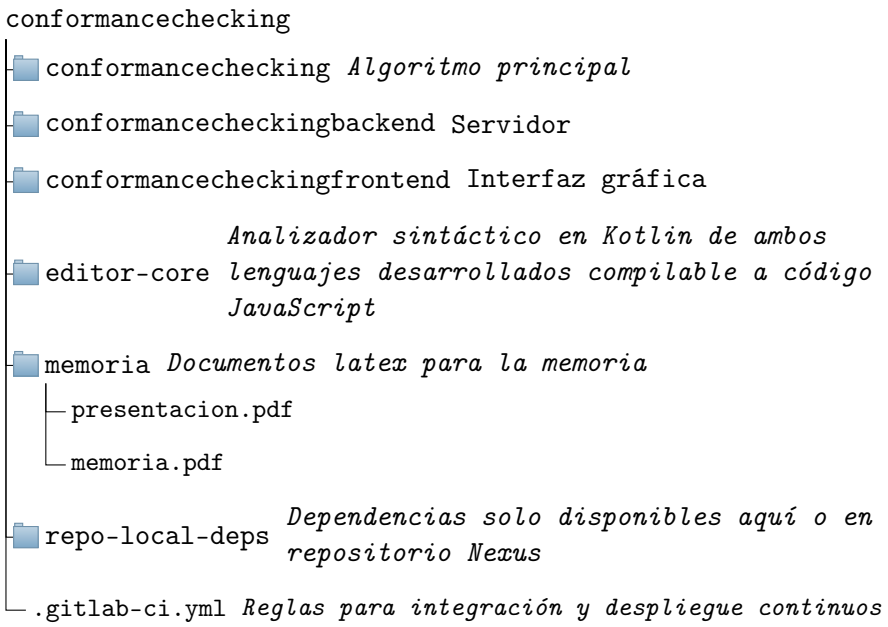


Figura 3.3: Estructura del proyecto.

3.4.5. Identificación de elementos de gestión de la configuración

Para los nombres de los documentos no es necesario seguir una notación determinada ya que la información sobre fecha de creación, descripción del cambio asociado y demás la proporcionar el repositorio (ver Subsección 3.4.3). Los elementos de gestión de la configuración de este proyecto son:

- Código algoritmo.

- Código servidor.
- Código interfaz gráfica.
- Memoria (alcance, análisis, catálogo, especificación y estimación de requisitos, planificación de los interesados, diagramas de diseño, EDT, cronograma...).
- Documentación (manuales y otros documentos).
- Presentación.

3.4.6. Control de la gestión de configuración

En esta actividad se lleva a cabo la creación de la estructura del repositorio en GitLab y la identificación de las peticiones de cambio. Para más información sobre las herramientas usadas y la forma de uso se puede ver la Subsección 3.4.2. La tarea de crear el repositorio es del rol dueño del mismo. Todos los cambios aparecerán reflejados en el repositorio. Esto es responsabilidad del gestor de cambios.

Es posible que surjan cambios en los que no sea necesario una reunión con los tutores. No obstante, existen ciertos cambios que sí requieren una reunión de grupo para tomar la decisión de si realizarlo o no. Algún ejemplo de este tipo de cambios son los cambios de concepto, relacionados con la comprensión del proyecto por parte de los miembros, o los cambios en la estructura del proyecto, ya sean de un apartado concreto de la documentación o de remodelación de contenidos del mismo.

En el caso del código, memoria y otra documentación, se usarán comentarios cuando sea necesario para marcar cambios y correcciones que deben hacerse. Estos comentarios comienzan con:

TODO. Describe una funcionalidad o cambio que puede ser implementado para mejorar la calidad pero no sería necesario.

FIXME. Indica que una característica contiene algún error o lo que debería ser revisado.

Al hacer el cambio, se eliminará este comentario. El IDE utilizado proporciona una lista de todos los comentarios del proyecto que contienen estas marcas para ver rápidamente los cambios identificados a realizar.

3.4.7. Estado de los elementos de la configuración

Esta actividad es la encargada de actualizar los elementos de gestión de configuración y el control de cuando un documento entra o sale de la línea base. Para establecer las líneas base en el proyecto tiene que haber un diagrama de hitos donde indico las líneas base y los diferentes entregables que tiene cada una de las líneas base visible en Sección 3.6.

Durante el desarrollo del proyecto, se trabaja con diversos documentos que sufren cambios a medida que se avanza en el producto. Cuando un documento entra en línea base, pasa a estar bajo los efectos de la gestión de la configuración. El manejo de los ECS es responsabilidad del gestor de configuración.

Para llevar una correcta gestión del estado de los elementos de la configuración es conveniente emplear un repositorio en donde se organizan los documentos de forma ordenada. Se emplea el repositorio en el que se pueden diferenciar subcarpetas destinadas a los documentos preparados para entrega (memoria) y otras destinadas a los proyectos. La gestión del repositorio es responsabilidad del dueño del mismo y se puede comprobar su estructura en la Figura 3.3. Para más información sobre cómo se gestionan los elementos de configuración en este proyecto se puede consultar [26].

3.5. Metodología

Se escogió un ciclo de vida por incrementos porque es el modelo que más se adecuaba para este proyecto. La justificación por la que no se escogieron otros ciclos de vida es la siguiente:

- Un ciclo de vida en cascada implicaría demasiado riesgo. Usar nuevas tecnologías lleva a errores, que podrían provocar fácilmente que se necesite reiniciar el proyecto desde el análisis, superando los pocos recursos disponibles por sólo un error grave.

- Usar un ciclo de vida de prototipos podría reducir la calidad del software final por falta de tiempo y la necesidad de usar un prototipo como la versión final del software, mientras que en el ciclo de vida por incrementos se establecen unos límites de tiempo para cada incremento y al final de estos el incremento debe estar completo y probado.

- El modelo en espiral se centra demasiado en los riesgos, lo que podría ser demasiado costoso para un proyecto como este y puede ser mayor que el costo de la construcción del sistema.

- No considero que haya suficientes interacciones con los tutores (11,25 horas) como para desarrollar un ciclo de vida ágil, en el que se dependería en gran medida de estos. Además, programación extrema es recomendable emplearla sólo en proyectos a corto plazo.

Como se tienen los objetivos claros, con los recursos conocidos se puede construir exactamente la aplicación pensada, aún así confirmando en cada incremento que las funcionalidades implementadas son correctas para un riesgo mínimo.

Además, se toman ideas de los ciclos de vida ágiles en ciertos temas para acelerar aún más el desarrollo del trabajo. De programación extrema se usan

las técnicas de integración continua y en ciertos casos *Test Driven Development* (TDD) para asegurar cierta calidad del código.

Concretamente, esta metodología está basada en realizar una serie de iteraciones. Al final de cada una de ellas se puede mostrar el progreso ya que se han creado y probado funcionalidades nuevas. Esto permite dar a los tutores una mejor idea de la dirección del proyecto, y les resulta más fácil explicar sus necesidades. Al enfocar las iteraciones a conjuntos de funcionalidades a implementar, se acaba construyendo un sistema más modular, evitando crear un sistema monolítico difícil de mantener.

3.6. Planificación

En esta sección se detalla la planificación temporal realizada para el proyecto que se está desarrollando. Se comienza analizando el objeto a medir, con el fin de identificar las tareas de cada fase.

Como se ha explicado en la Sección 3.5, la metodología aplicada en este proyecto es por incrementos. Esta caracteriza su ciclo de vida por la existencia de iteraciones en la que se construyen partes completamente funcionales del software ampliamente probados que se despliegan para que el cliente pueda comprobar el progreso.

Para la realización de dicha identificación de tareas se ha utilizado una técnica muy usada en la gestión de proyectos, la Estructura de Desglose del Trabajo o EDT, que da una visualización general del proyecto sin detallar información sobre las tareas como fechas ni la paralelización de las mismas. Un EDT consiste en una descomposición jerárquica que representa el trabajo requerido para completar un proyecto. Está disponible en la Figura 3.4.

El proyecto comienza con una fase de formación y estudio de las tecnologías. También es necesario una preparación del clúster del servidor en el que se ejecutará el código opcionalmente (esta tarea podría realizarse en un punto posterior del proyecto). La primera iteración consiste en el desarrollo del algoritmo principal y de todos los añadidos indicados como las restricciones, la carga, adaptación y exportación de registros y consultas, que se detallarán en el capítulo del incremento. El segundo incremento es el desarrollo del servidor web, utilizable por desarrolladores de forma remota, que debe manejar todas las funcionalidades de forma eficiente, con privacidad por usuario, informando del progreso al usuario y el último se encarga de implementar la interfaz gráfica de forma que los usuarios sean capaces de utilizar toda la aplicación. Después se genera toda la documentación, se realiza la entrega y la presentación.

En las iteraciones en este proyecto se desarrollan módulos completos que proporcionan un conjunto de funcionalidades. Por este mismo motivo, solo se realizan tres incrementos, buscando que sean lo más independientes posible: la

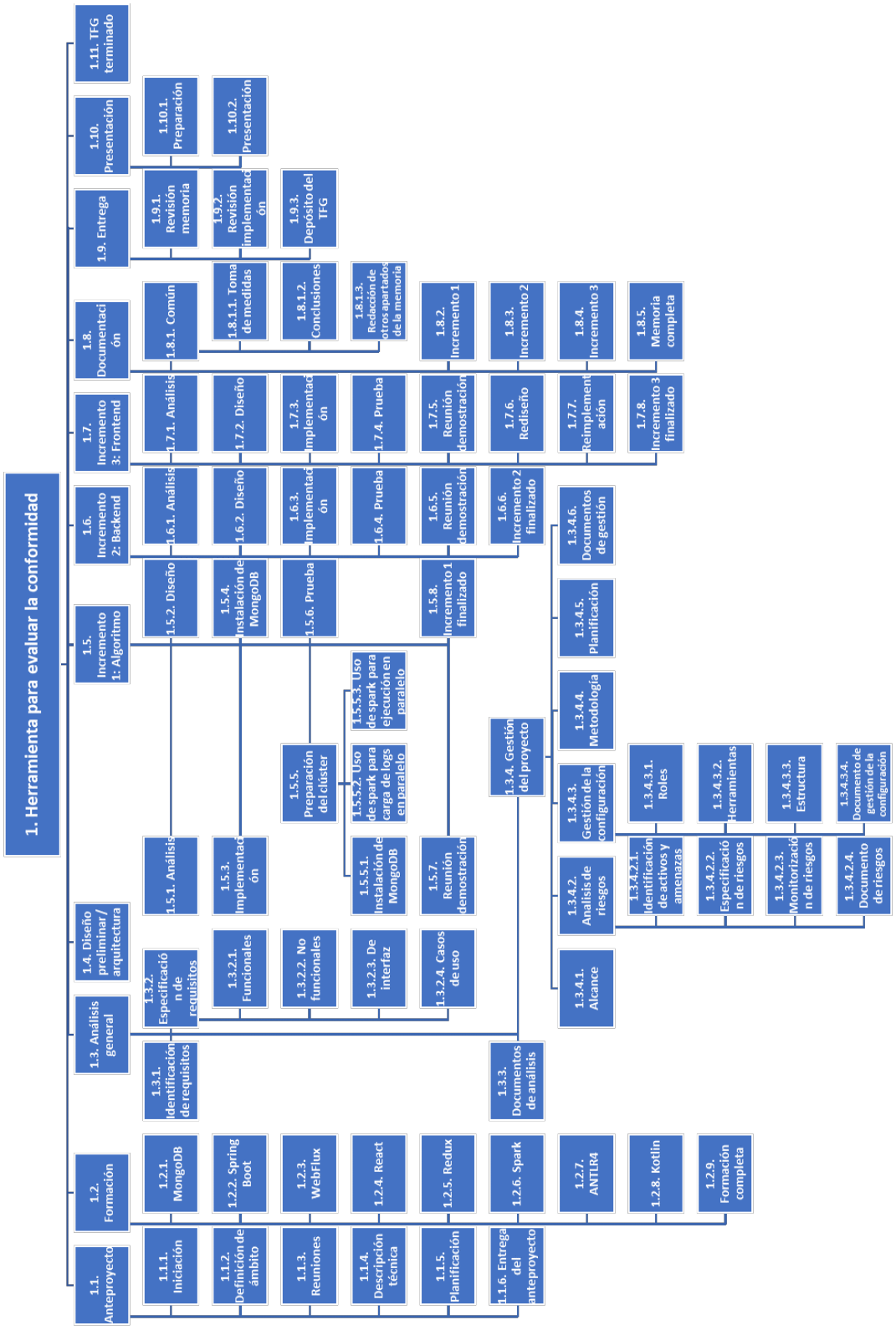


Figura 3.4: Estructura de Desglose del Trabajo del proyecto.

biblioteca que proporciona las funcionalidades principales y puede ser usado como biblioteca; el servicio web que permite acceder a ellas remotamente y gestiona usuarios, registros y otros recursos; y la interfaz web, que está orientada a permitir que los usuarios puedan usar la aplicación fácilmente.

El total de trabajo son las 14 semanas descritas en la planificación del anteproyecto, sumando aproximadamente 420 horas. El proyecto se configuró para 30 horas semanales como indica el anteproyecto. Ahora que se dio una idea de la estructura del proyecto, se incluye más detallada la planificación a modo de diagramas de *Gantt* o cronogramas.

Para poder ver todos los datos, el cronograma se dividió por incrementos. Primero se muestra la jerarquía de mayor nivel como *Gantt*, para dar una idea de la estructura. Esto se puede ver en la Figura 3.5. Ya en esta figura se puede indicar que se trata de los resultados de la planificación tras la realización del proyecto, es decir, en el diagrama la línea base marca la planificación inicial y se sufrieron algunos cambios, como se puede ver en las tareas cercanas a la fecha actual (línea verde).

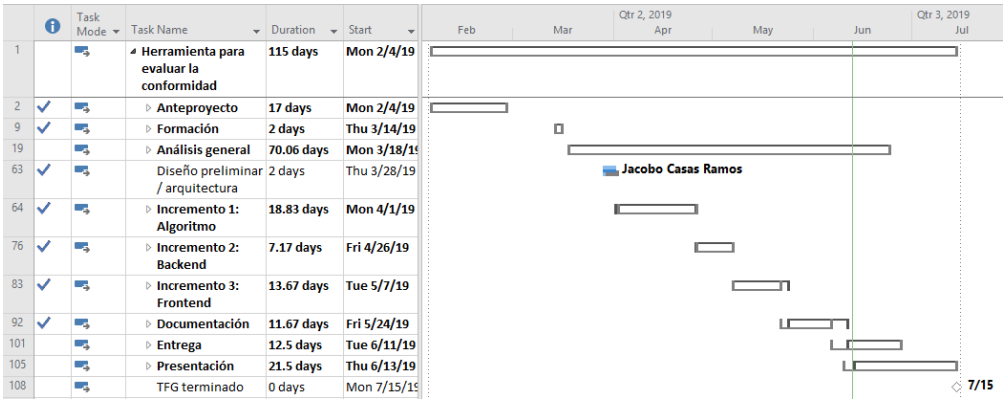
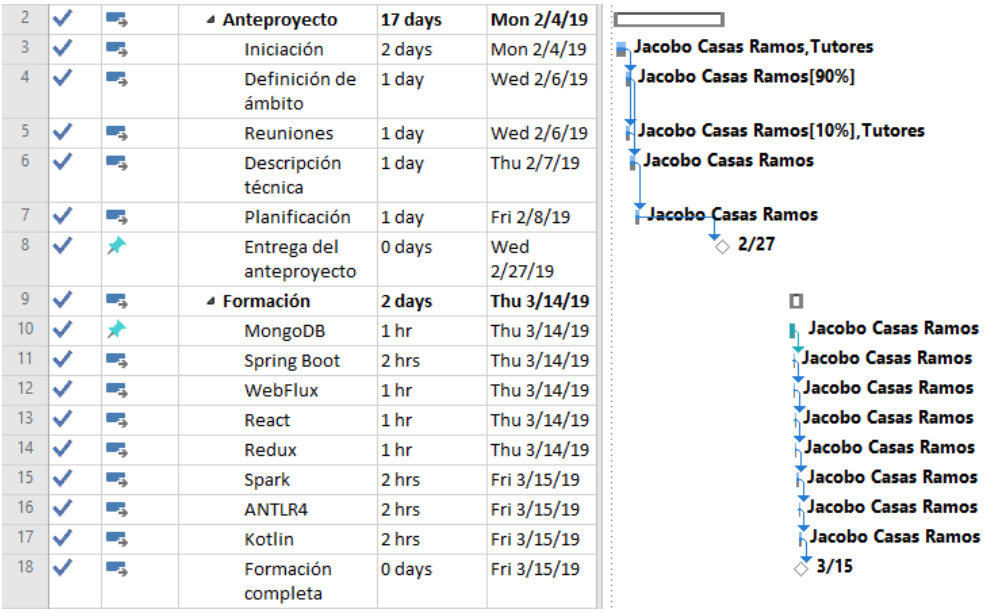


Figura 3.5: Resumen *Gantt*.

3.6.1. Anteproyecto y formación

En la Figura 3.6 se incluye el trabajo realizado para el anteproyecto y el trabajo de formación previo al comienzo del TFG. La formación se comienza un tiempo después de terminar el anteproyecto porque se realizan 30 horas semanales.

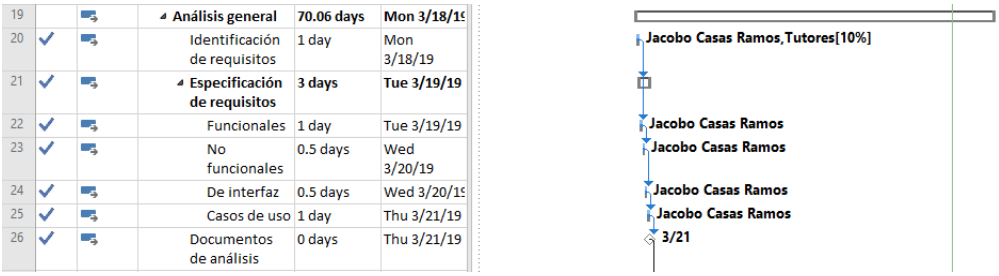


2	✓		**Anteproyecto**	**17 days**	**Mon 2/4/19**
3	✓		Iniciación	2 days	Mon 2/4/19
4	✓		Definición de ámbito	1 day	Wed 2/6/19
5	✓		Reuniones	1 day	Wed 2/6/19
6	✓		Descripción técnica	1 day	Thu 2/7/19
7	✓		Planificación	1 day	Fri 2/8/19
8	✓		Entrega del anteproyecto	0 days	Wed 2/27/19
9	✓		**Formación**	**2 days**	**Thu 3/14/19**
10	✓		MongoDB	1 hr	Thu 3/14/19
11	✓		Spring Boot	2 hrs	Thu 3/14/19
12	✓		WebFlux	1 hr	Thu 3/14/19
13	✓		React	1 hr	Thu 3/14/19
14	✓		Redux	1 hr	Thu 3/14/19
15	✓		Spark	2 hrs	Fri 3/15/19
16	✓		ANTLR4	2 hrs	Fri 3/15/19
17	✓		Kotlin	2 hrs	Fri 3/15/19
18	✓		Formación completa	0 days	Fri 3/15/19

Figura 3.6: *Gantt*: Anteproyecto y formación

3.6.2. Análisis general

En la Figura 3.7 se incluye parte del trabajo realizado para el análisis inicial, el cual se realizó según lo previsto y no hubo cambios visibles en la línea base.



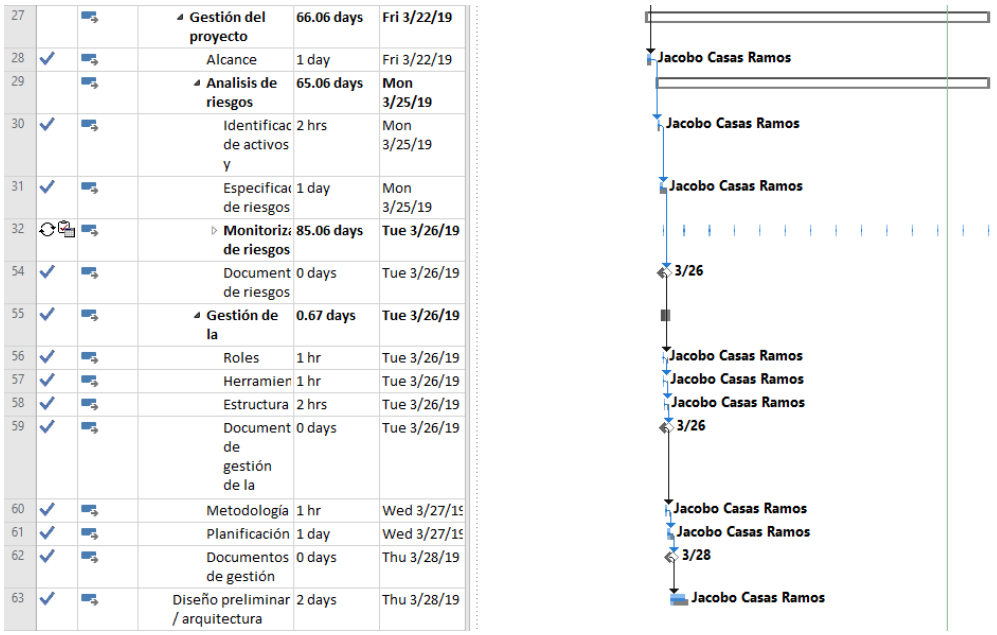
19			**Análisis general**	**70.06 days**	**Mon 3/18/19**
20	✓		Identificación de requisitos	1 day	Mon 3/18/19
21	✓		**Especificación de requisitos**	**3 days**	**Tue 3/19/19**
22	✓		Funcionales	1 day	Tue 3/19/19
23	✓		No funcionales	0.5 days	Wed 3/20/19
24	✓		De interfaz	0.5 days	Wed 3/20/19
25	✓		Casos de uso	1 day	Thu 3/21/19
26	✓		Documentos de análisis	0 days	Thu 3/21/19

Figura 3.7: *Gantt*: Análisis general 1

En la Figura 3.8 se incluye la segunda parte trabajo realizado para el análisis inicial y el diseño de la arquitectura del sistema, el cual se realizó según lo previsto y no hubo cambios en la línea base. La tarea 55 es gestión de la configuración.

La gestión de riesgos se realizó más rápido de lo planificado (línea base), acelerando el proyecto.

Existe la tarea de motorización de riesgos que es periódica ya que el proceso es continuo según los indicadores y las medidas indicadas en la Sección 3.3.

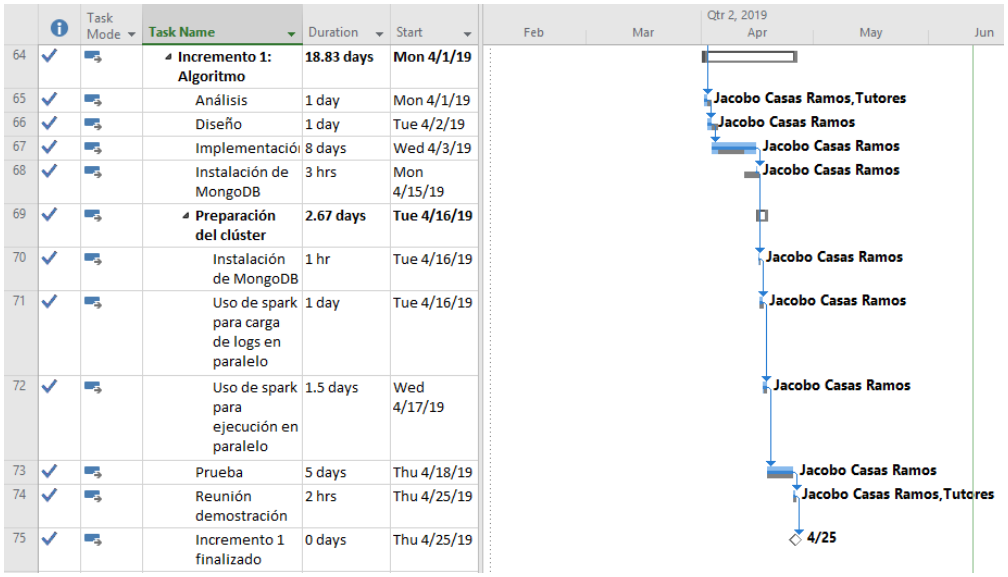


			Task Name	Duration	Start
27			**Gestión del proyecto**	**66.06 days**	**Fri 3/22/19**
28	✓		Alcance	1 day	Fri 3/22/19
29			**Analisis de riesgos**	**65.06 days**	**Mon 3/25/19**
30	✓		Identificac de activos y	2 hrs	Mon 3/25/19
31	✓		Especificac de riesgos	1 day	Mon 3/25/19
32			**Monitoriz de riesgos**	**85.06 days**	**Tue 3/26/19**
54	✓		Document de riesgos	0 days	Tue 3/26/19
55	✓		**Gestión de la**	**0.67 days**	**Tue 3/26/19**
56	✓		Roles	1 hr	Tue 3/26/19
57	✓		Herramien	1 hr	Tue 3/26/19
58	✓		Estructura	2 hrs	Tue 3/26/19
59	✓		Document de gestión de la	0 days	Tue 3/26/19
60	✓		Metodología	1 hr	Wed 3/27/19
61	✓		Planificación	1 day	Wed 3/27/19
62	✓		Documentos de gestión	0 days	Thu 3/28/19
63	✓		Diseño preliminar / arquitectura	2 days	Thu 3/28/19

Figura 3.8: *Gantt*: Análisis general 2 y diseño

3.6.3. Incremento 1

En la Figura 3.9 se incluye el trabajo realizado para el incremento 1, que consiste en un ciclo en cascada completo para implementar el algoritmo principal, ya que se usa el ciclo de vida por incrementos.



			Task Name	Duration	Start
64	✓		**Incremento 1: Algoritmo**	**18.83 days**	**Mon 4/1/19**
65	✓		Análisis	1 day	Mon 4/1/19
66	✓		Diseño	1 day	Tue 4/2/19
67	✓		Implementació	8 days	Wed 4/3/19
68	✓		Instalación de MongoDB	3 hrs	Mon 4/15/19
69	✓		**Preparación del clúster**	**2.67 days**	**Tue 4/16/19**
70	✓		Instalación de MongoDB	1 hr	Tue 4/16/19
71	✓		Uso de spark para carga de logs en paralelo	1 day	Tue 4/16/19
72	✓		Uso de spark para ejecución en paralelo	1.5 days	Wed 4/17/19
73	✓		Prueba	5 days	Thu 4/18/19
74	✓		Reunión demostración	2 hrs	Thu 4/25/19
75	✓		Incremento 1 finalizado	0 days	Thu 4/25/19

Figura 3.9: *Gantt*: Incremento 1

3.6.4. Incrementos 2 y 3

En la Figura 3.10 se incluye el trabajo realizado para el incremento 2, que consiste en el desarrollo del servidor; y del incremento 3, que consiste en el desarrollo del interfaz gráfica.

En estos destaca un retraso experimentado en el desarrollo de la interfaz gráfica, porque los tutores recomendaron un cambio importante en el diseño ya implementado. Este cambio se describirá en detalle en el apartado adecuado. Se decidió permitir al usuario usar ambas interfaces implementadas con un sencillo interruptor en la página, usando por defecto la nueva interfaz. Se disponía de un colchón de tiempo al final del proyecto que se vió afectado por este cambio.

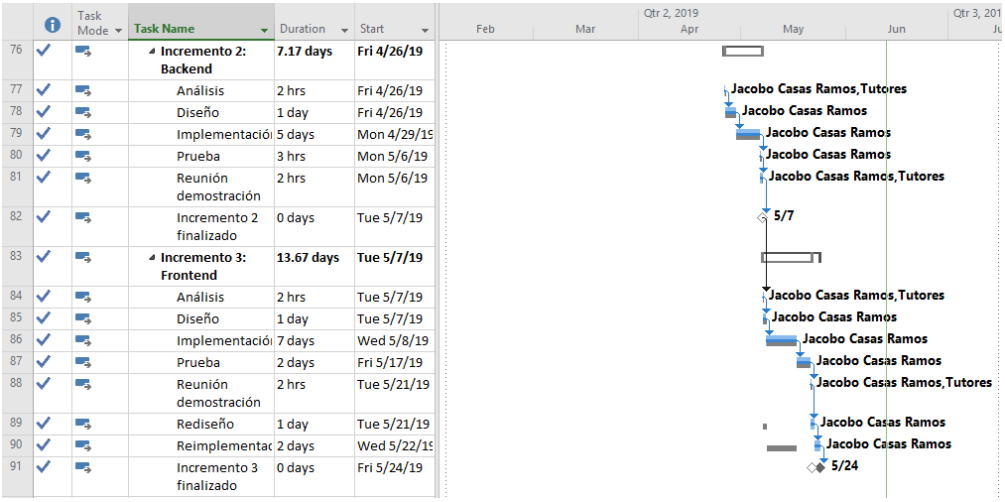


Figura 3.10: *Gantt*: Incrementos 2 y 3

3.6.5. Documentación y finalización del proyecto

En la Figura 3.11 se incluye el trabajo realizado para documentar y finalizar el proyecto, lo cual implica tomar medidas de rendimiento, escribir la memoria de los incrementos, conclusiones y otros apartados de la memoria restantes, revisarla, revisar el código, imprimir la memoria, preparar el CD y trabajar en la presentación.

Cabe mencionar que, como se puede ver en el *gantt*, en la actualidad se está trabajando en la revisión de la memoria con pocos días de retraso. El progreso se indica con la barra azul oscuro fina de progreso en el medio de las tareas.

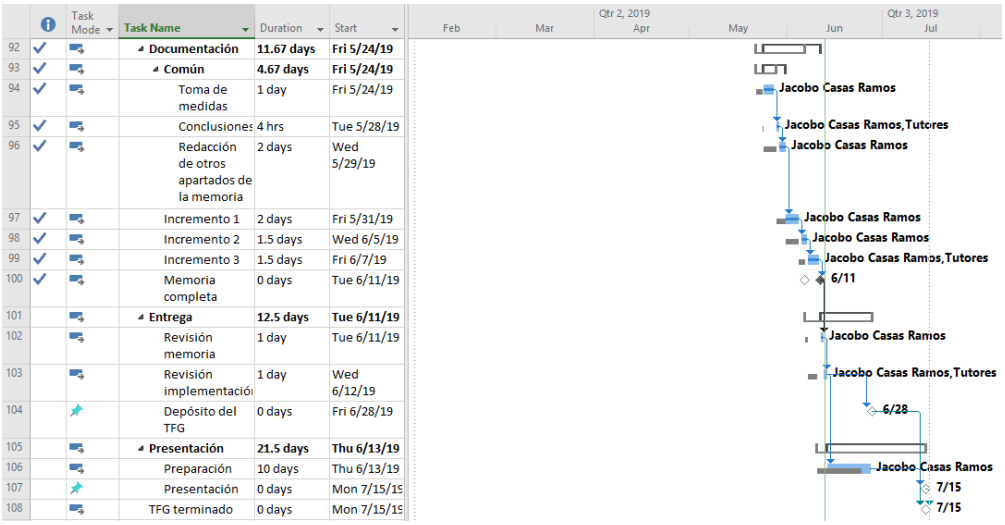


Figura 3.11: *Gantt*: Finalización del proyecto

3.7. Presupuestos

Los presupuestos de todo el proyecto no resultan complejos ni varían entre incrementos, por lo que los incluyo en este apartado compartido. Para este proyecto los costes se clasifican según sean directos o indirectos. Esto significa:

Directos: son los que de forma clara pertenecen al proyecto, como por ejemplo las horas trabajadas por un empleado o la amortización de equipos.

Indirectos: son compartidos por diferentes proyectos (no se pueden asignar a uno en concreto) de forma que su cálculo es más difícil, como por ejemplo incluiría el coste del material de oficina, alquiler de instalaciones, consumo eléctrico o el servicio de limpieza.

Para que el proyecto pueda resultar útil para trabajos futuros, puede ser necesario un cálculo de costes aunque en este caso se trate de un proyecto académico en el que los costes serían casi todos indirectos. Para evitar esto y dar resultados más reales, se tratan las horas de trabajo como costes directos suponiendo un salario. Según los resultados obtenidos por The Standish Group como se nos indicó en Ingeniería del Software: "The Standish Group research shows a staggering 31.1 % of projects will be cancelled before they ever get completed. Further results indicate 52.7 % of projects will cost 189 % of their original estimates" y "On the success side, the average is only 16.2 % for software projects that are completed on- time and on-budget"[4].

3.7.1. Costes directos

Como ya se indicó en el apartado anterior, para los costes directos se supone que las horas trabajadas tienen un salario asociado para que los resultados

[4] *The Standish Group Report: CHAOS*, p. 3 [2].

Año	Mes	DT	RBM (€)	BC (€)	SB (€)	PE (€)	CP (€)	Liq. (€)
2019	marzo	17	1067,07	682,70	585,17	0,00	222,56	22,88
2019	abril	30	1067,07	1256,70	1067,07	0,00	408,62	40,37
2019	mayo	31	1067,07	1256,70	1067,07	0,00	408,62	41,71
2019	junio	28	1067,07	1161,92	995,93	620,58	378,79	37,68

Tabla 3.51: Costes por mes

Año	B (€)	PE (€)	Liq. (€)	CC (€)	SS (€)	TOTAL (€)
2019	3.715,24	620,58	142,64	4.478,46	1.418,59	5.897,05

Tabla 3.53: Costes totales

sean más realistas. Para identificar el salario se toma como referencia el salario mínimo de un investigador en el CiTIUS que está realizando el proyecto, este lo indica la calculadora de contratos de la USC: "Segundo a Normativa da Universidade de Santiago de Compostela para a contratación de persoal con cargo a Actividades de Investigación, Desenvolvemento e Innovación (I+D+I) o contrato actual ten un salario mínimo anual de 17.428,80 €, salario mínimo mensual de 1.244,91 €. Para a xornada laboral parcial de 30,00 horas semanais o salario mínimo é de 1.067,07 €"[5] [4].

Entonces los resultados por mes están en la Tabla 3.51 y los totales están en la Tabla 3.53. En estas tablas, DT significa Días Trabajados, RBM es la Retribución Bruta Mensual, BC es la Base de Cotización, SB es el salario bruto, PE son las pagas extras, CP es la Cota Patronal, SS es la Seguridad Social, B es Bruto, ST es el salario del trabajador, CC es Coste Contrato y Liq. es la liquidación. De la misma forma se calculan los costes directos por salarios de los tutores de proyecto, pero en este caso se considera que los tutores son clientes y por lo tanto no contribuyen a sus horas dedicadas a los costes del proyecto. Serían asignados durante 11,25 horas cada uno durante el proyecto, lo que con un salario mensual mínimo de 1819,39 €[6] [4] resultaría en 479,74 € de coste para cada uno.

Además, existen costes directos por la compra del CD para la entrega de la memoria. No para el grabado del mismo porque se dispone de una grabadora de CDs. También se suma el coste de imprimir los documentos finales.

[5] *La calculadora se configuró para un ingeniero graduado trabajando como investigador en formación para realizar el proyecto trabajando 30 horas a la semana.*

[6] *La calculadora se configuró para un doctor trabajando como investigador asociado para realizar el proyecto trabajando 11,25 horas a la semana durante una semana.*

Item	Coste (€)	Vida útil	Uso	Amortización (€)
Ordenador	1800,00	6 años	14 semanas	$\frac{1800,00*14*7}{6*365} = 80$
Clúster CiTIUS	11764,78	10 años	4 días	$\frac{11764,78*4}{10*365} = 12,89$
Estación *dock*	89,99	6 años	14 semanas	$\frac{89,99*14*7}{6*365} = 4,03$

Tabla 3.55: Amortizaciones

También se considera la amortización del equipo usado para desarrollar el proyecto como un coste directo, y se incluye también la estación *dock*[7] comprada para poder conectar el portátil a la pantalla externa y facilitar el desarrollo del trabajo. Se considera que ambos tienen una vida útil de 6 años y se usan las 14 semanas para este proyecto, para poder hacer los cálculos de amortización. Los resultados se muestran en la Tabla 3.55. La pantalla que se usó para realizar el proyecto se considerá que ya ha sido amortizada porque llevaba muchos años de uso en distintos proyectos para los que ya se tendría en cuenta la amortización. Además, fue necesario el uso del clúster de *Big Data* del CiTIUS para realizar pruebas usando *Spark* con distintas tecnologías y escoger la más adecuada. La columna uso indica el uso en el proyecto.

Todo el software que usa el proyecto es gratuito o usa licencias de prueba (o licencia de estudiante en el caso del *Intellij Ultimate*), por lo que no añade ningún coste directo. Entonces el resumen final de costes directos se ve en la Tabla 3.57, donde A es amortización, y los costes directos del proyecto ascienden a 6.068,97€.

ST (€)	A (€)	CD (€)	Impresión (€)	TOTAL (€)
5.897,05	96,92	5,00	70,00	6.068,97€

Tabla 3.57: Resumen de costes directos

[7]Más información: https://az31609.vo.msecnd.net/literature/337370c6-6b1b-4d31-ad48-2c48ed206bda.pdf

3.7.2. Costes indirectos

En el caso de este proyecto incluye el coste de electricidad, acceso a internet y materiales necesarios para completarlo, como el cluster de Big Data del CiTIUS. Los costes indirectos por facultad se indican en el documento de la USC indicado en la bibliografía que indica que son del 21 % para el departamento de electrónica y computación, que es al que está asociado el TFG[8].

Los costes indirectos entonces ascienden al 21 % de 6.068,97: 1.274,48€.

3.7.3. Resumen de costes

Los costes finales se pueden ver en la Tabla 3.59.

Directos (€)	Indirectos (€)	TOTAL
6.068,97	1.274,48	**7.343,45€**

Tabla 3.59: Costes del proyecto

[8] *Costes Indirectos en Proyectos de I+D de Universidades - USC*, p. 9 [3].

Capítulo 4

Arquitectura

Este capítulo es un resumen, que describe la forma en la que interaccionan los incrementos. Entonces se incluye un primer esquema de la arquitectura del sistema ya que la arquitectura detallada se realizará en cada incremento como indica la planificación, centrándose en las funcionalidades de la iteración.

Este diagrama de arquitectura se puede ver en la Figura 4.1. Está muy simplificado ya que la comunicación también incluye usuarios, exportación de registros y consultas... El cliente (usuario) accede mediante una API REST especificada mediante *OpenAPI* [25] a un servidor que es capaz de ejecutar el algoritmo en una configuración sencilla sobre sí mismo, o sobre un clúster externo. Este acceso puede realizarse mediante la interfaz gráfica web o directamente utilizando la API. Cada nodo del clúster ejecuta el algoritmo sobre las trazas asignadas al nodo del registro que se estudia, utilizando *Spark* sobre la plataforma *yarn* (que maneja los recursos y planifica el trabajo de los nodos del clúster) con almacenamiento en el sistema de ficheros distribuido *HDFS*. El trabajo se paraleliza en función de la traza ya que casi no hay dependencias ni necesidad de sincronizarse entre nodos la procesarlas salvo para iniciarse y agregar los resultados, y en un entorno *Big Data* se reparte adecuadamente el trabajo al disponer de muchas trazas. Los registros se leen de una base de datos *MongoDB* que en el caso óptimo está distribuida entre todos los nodos que trabajan para maximizar la localidad de los datos. Los resultados se sincronizan entre todos los nodos y se devuelven al servicio, que informa al usuario de estos.

El servidor y el clúster no tienen por qué ser entidades separadas, sino que se puede ejecutar el algoritmo dentro del propio servidor en una configuración más sencilla, que es la que se usa por defecto. Sin embargo, la posibilidad de mantener el clúster externo permite no tener que desplegar al servidor en todos los nodos y aprovechar más potencia del clúster para el algoritmo, mientras que el servidor sigue realizando tareas como gestión de usuarios, carga de registros, etc. que deben realizarse secuencialmente. Pueden separarse dos instancias de *MongoDB*: con la que trabaja el servidor que guarda información sobre los usuarios, metadatos sobre registros y consultas, restricciones y resultados; y con la que trabaja el clúster que mantiene los registros y las consultas. Estas

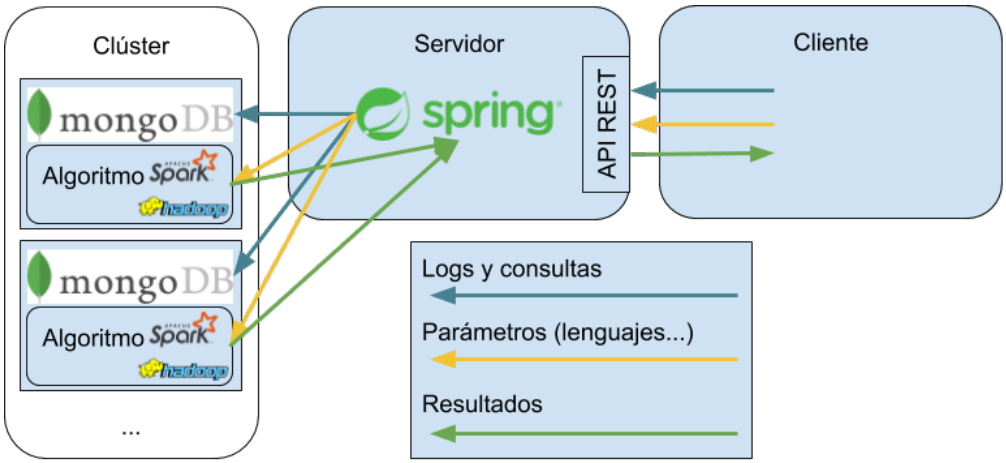


Figura 4.1: Diagrama de arquitectura del proyecto.

también están configuradas por defecto para que sean dos bases de datos de la instancia de *MongoDB* desplegada localmente en el servidor, pero se pueden configurar como en la imagen.

Para que esta arquitectura pueda trabajar con registros de gran tamaño (*Big Data*), el cuello de botella podría ser el servidor si debe cargar todo el registro en memoria al importarlo antes de guardarlo en la base de datos *NoSQL*. Para conseguir esto se modificará la biblioteca base, que necesita cargar los registros completamente en memoria antes de trabajar con ellos. De esta forma, cada traza se emite a la base de datos como un documento separado cada vez que se lee y procesa, antes de leer la siguiente traza, y la arquitectura permite trabajar con registros tan grandes como permita el almacenamiento de la base de datos, la cual se puede distribuir mediante *sharding*[1] para aumentar la capacidad, el rendimiento y la disponibilidad fácilmente.

En capítulos siguientes se describirá con más detalle la arquitectura de cada incremento entre otros apartados. Concretamente en el capítulo siguiente se describe el algoritmo, que se ejecutaría en los nodos del clúster o en el servidor; en el siguiente se desarrolla el servidor, que daría acceso a toda esta arquitectura, pero solo a desarrolladores; y en el último incremento se desarrollará la interfaz gráfica, que permitirá que el cliente sea un usuario que pueda trabajar de forma sencilla con la herramienta.

[1] Es una forma de escalado horizontal implementada en *MongoDB*.

Capítulo 5

Incremento 1: Algoritmo

En este incremento se realiza la implementación completa del algoritmo y de las utilidades que permiten su uso, como la importación y exportación de datos. En resumen, se hace la implementación completa de la biblioteca que será usada posteriormente por los miembros de inVerbis. A modo de introducción se detallan más profundamente las tareas del algoritmo.

El algoritmo esta basado en la idea de alineamientos, usando una implementación de *token replay* para ejecutar los eventos de la traza en el modelo e ir comprobando si la traza se adapta adecuadamente al modelo. La ejecución basada en marcas hace coincidir una traza y un modelo de red de Petri, comenzando desde el lugar inicial, para descubrir qué transiciones se ejecutan y en qué lugares tenemos tokens restantes o faltantes para la instancia de proceso dada. De esta forma es útil para realizar comprobación de la conformidad tomando como válidas las trazas en las que se llega al lugar final del modelo usado como consulta. Además, existen dos modos de consulta: *estricto*, en el que la traza debe adaptarse completamente al modelo, es decir, no deben existir movimientos ni en el modelo ni en la traza, y *relajado*, en el que se permiten movimientos que únicamente tienen lugar en la traza (eventos que no tiene importancia si ocurren o no ya que no son estudiados en la consulta del usuario).

Además, se permite definir un *fitness* mínimo que por defecto se mantiene al 100 %. Este permite que también existan ciertos movimientos solo en el modelo, pero estos tienen un coste. El *fitness* en los pasos del algoritmo se calcula dividiendo este coste por el peor coste posible (tanto por uno). El peor coste posible se calcula estudiando el mínimo número de pasos para completar la consulta, lo cual se puede obtener ejecutando el algoritmo comprobando la traza vacía sobre el modelo de consulta. Por ejemplo, el modelo formado por la secuencia de A, B y C tiene un coste mínimo de 3. Si se procesa la traza A, B, está no se aceptará salvo que el *fitness* mínimo sea inferior al 66 % ya que en ese caso se permitiría realizar el movimiento solo en el modelo de la actividad C, confirmando que la traza se adapta parcialmente a la consulta definida; mientras que si se procesa A, D, la diferencia con el modelo sería demasiada y no se aceptaría la traza. Entonces la utilidad de este modo es permitir un modo aún

más relajado de comprobación de consultas. El coste de cada paso del algoritmo es completamente configurable mediante la implementación de restricciones, lo cual permite definir consultas aún más complejas.

Todos los modos permite realizar consultas con condiciones sobre determinadas actividades, independientemente de si se han registrado otras actividades entre ellas. En estos casos, el registro de eventos contiene más información que la que desea consultar el usuario.

Para realizar las consultas sobre los modelos se implementan dos lenguajes: uno que permite especificar la *estructura del modelo*, que está basado en el modelado de procesos a través de árboles (*process trees*); y otro que permite definir las condiciones que restringen la exploración del modelo a la hora de realizar consultas. Más específicamente:

- Lenguaje de consulta: está formado por las funciones SEQ, LOOP, SWITCH y FORK, que representan una secuencia de actividades, un bucle del contenido, un OR de sus contenidos y un AND de sus contenidos respectivamente. Este lenguaje se puede ver claramente en el ejemplo de la Tabla 5.2[1]. En esta se muestra una representación *causal net*, la cual es usada internamente por el algoritmo y es fácil de interpretar. La representación es <(I)nput/(O)utput>(Actividad_#)=[{Act., Act...}, {Act...}...], en la que primero se indica si se están listando las entradas o salidas de la actividad entre paréntesis. Después del igual se indican entre llaves los grupos de actividades entre los cuales se aplica un OR (SWITCH), y entre estos grupos se aplica un AND (FORK).

- Lenguaje de condiciones: se basa en el lenguaje de una calculadora que tiene acceso a tiempos y KPIs asociados a los eventos de la traza, y que se usarán para determinar el coste de todos los pasos del algoritmo recuperando los datos que necesite cuando estén disponibles. Un ejemplo de consulta sería $((A[\text{“}tiempo''] > B[\text{“}tiempo''])\ AND\ (C[\text{“}users''] < 2000))$, que consiste en que el tiempo del último evento A sea posterior al tiempo de B o el número de usuarios guardado en el evento C sea inferior a 2000. Si esta condición se evalúa como cierta en un paso, se permite que siga el algoritmo, y en otro caso el algoritmo debe buscar otro camino. Si el lenguaje devuelve un entero o flotante en lugar de booleano, se interpreta como el coste del paso, lo que se puede usar con el *fitness* mínimo indicado anteriormente. En la Tabla 5.3 se incluye el léxico disponible en este lenguaje, sobre el que se construyen las reglas de análisis sintáctico.

Los resultados del algoritmo son el número de trazas y de ocurrencias válidas (puede haber más de una ocurrencia en una traza si el modelo se cumple más de una vez en la traza). Además, de forma opcional, se pueden guardan los IDs

[1]En la representación gráfica, los SWITCH y FORK se representan de la misma forma, pero como se puede ver en la representación de *causal net*, la estructura interna los contempla correctamente.

Código fuente	***Causal net***	
SEQ(A, FORK(B, SWITCH(C, LOOP(D))), E)	I(E_4)=[{1}, {3,2}] O(E_4)=[] I(D_3)=[{3}, {0}] O(D_3)=[{3}, {4}] I(C_2)=[{0}]	O(C_2)=[{4}] I(B_1)=[{0}] O(B_1)=[{4}] I(A_0)=[] O(A_0)=[{1}, {3,2}]

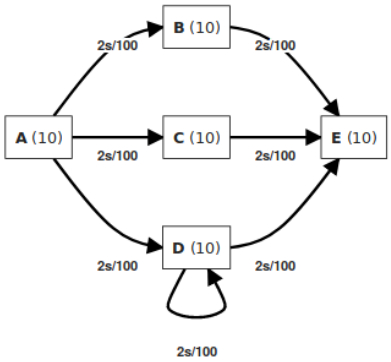


Tabla 5.2: Ejemplo de lenguaje de consultas.

de las trazas válidas para realizar consultas posteriores sobre ellas, se puede recuperar el número de veces que el algoritmo paso por cada transición entre actividades del modelo, el tiempo medio de estas transiciones entre tareas y/o el tiempo medio de las tareas para todas las trazas.

5.1. Arquitectura

5.1.1. Descripción

El algoritmo de comprobación de la conformidad busca obtener el fitness (el número de trazas de un registro que pueden ser ejecutadas en un proceso) de cualquier proceso, pero el modelo no tiene por qué estar completo, es decir, el usuario puede definir parámetros que permiten un control sobre las trazas que admite el algoritmo no configurable solamente con el modelo. Los procesos que definen los usuarios no tendrían por qué adaptarse completamente a la traza (si así lo desean), ya que pueden querer aceptar trazas como A, B, C con un grafo que busca A $\rightarrow$ C y en el que no les importan las actividades que ocurran en el medio, algo que sería más difícil de comprobar con un descubrimiento automatizado de procesos.

Para aclarar la idea del *fitness* mínimo que maneja opcionalmente el algoritmo, si el algoritmo se configuró con un valor de *fitness* mínimo de 0.3, esto

COMMENT	RSQBRACKET	POINT
LINE_COMMENT	LBRACKET	POW
PREPROC	RBRACKET	PI
WS	LPAREN	EULER
NEWLINE	RPAREN	QUESTION
TYPE_INT	PLUS	COLON
TYPE_FLOAT	MINUS	NULL
TYPE_STRING	TIMES	DOUBLE
TYPE_BOOLEAN	DIV	INTEGER
COS	GT	STRING
SIN	LT	TEXT_OR_STRING
TAN	GE	BOOLEAN
ACOS	LE	TRUE
ASIN	EQ	FALSE
ATAN	DIFFERENT	SPECIAL_KEY_TRACE
LN	AND	SPECIAL_KEY_GLOBAL
LOG	OR	ERROR_CHAR
SQRT	NOT	
LSQBRACKET	COMMA	

Tabla 5.3: Lexemas disponibles en el lenguaje de condiciones.

significa que aceptará todas las trazas que cumplan el modelo en al menos un 30 % comparado con la longitud del camino mínimo a través de la consulta. La longitud del camino mínimo se calcula aplicando el algoritmo sobre una traza de cero eventos y sin limitación de *fitness* mínimo.

La aplicación de restricciones puede mejorar el rendimiento del algoritmo al permitir el filtrado de un mayor número de trazas en estados previos de ejecución a la vez que descartan trazas que realmente no interesan a los usuarios, y serían complejas de descartar de otras formas.

La base de datos usada para almacenamiento de registros será la base de datos MongoDB [8]. Se trata de una base de datos NoSQL, ya que los registros no tienen por qué tener una estructura predeterminada (algunos eventos pueden tener más campos que otros, como una compra que guarda el conjunto de productos mientras que un inicio de sesión no los guarda). Espero que una base de datos NoSQL permita un acceso más eficiente, y más opciones para manejar estas estructuras.

La implementación de este algoritmo debe permitir el uso de Spark [10] para paralelizar el procesamiento en un clúster. Este componente es necesario porque el procesamiento de registros puede llevar muchas horas y se trata de un proceso que se puede paralelizar porque no existen dependencias en el procesamiento de trazas diferentes dentro de un mismo registro.

De esta forma el algoritmo mejoraría su rendimiento en un clúster, y este resultaría escalable para ofrecer los servicios a más usuarios. Esto puede llevar asociado el uso de una base de datos externa para almacenar sobre todo los registros, aunque se podrían añadir también los modelos y restricciones para disponer de los datos en una base de datos desplegada en el propio clúster y mejorar significativamente el rendimiento del algoritmo. La información como los usuarios, nombres de registros, configuraciones de privacidad... seguirían estando disponibles en la base de datos del servidor, ya que sólo serían usados por su API pública. La base de datos secundaria para acceso desde el clúster es una configuración opcional para permitir un despliegue sencillo del servidor.

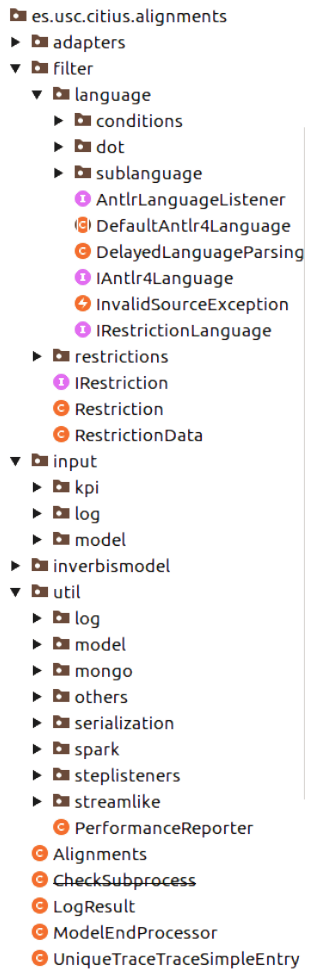


Figura 5.1: Estructura de paquetes.

Se desarrollará para el proyecto un lenguaje propio para definir procesos de uso muy fácil para los usuarios, aunque se permitirá la carga y exportación de modelos en otros formatos, al igual que se permitirá la carga y exportación de registros en múltiples formatos. También se desarrollará un lenguaje que permite expresar una gran variedad de condiciones aplicables sobre los datos asociados a los eventos (KPI), o a los tiempos de los eventos. Para estos lenguajes, además de las gramáticas e intérpretes, se implementarán distintas facilidades en la interfaz para la creación y edición de los mismos, ejecutados en la propia interfaz sin aumentar la carga del servidor para tareas como el análisis sintáctico para mostrar errores.

5.1.2. Diagrama y explicación de paquetes

Se comenzará con un diagrama de paquetes para ver la estructura del código. Este se puede ver en la Figura 5.2 y la Figura 5.3. La organización de estos se puede ver en la Figura 5.1. La herramienta utilizada para generar los diagramas es PlantUML [21] con representación gráfica mediante yFiles [23].

El paquete principal es *es.usc.citius.alignments*, el cual contiene el algoritmo y la clase de resultados entre otras. No se muestra en el diagrama de paquetes porque no lo permitía la herramienta al ser el padre de otros paquetes mostrados.

El paquete *adapters* contiene los adaptadores de registros de otros sistemas a este, proporcionando información de progreso y un sistema de caches en Java y almacenamiento en MongoDB automáticos.

El paquete *filter* se encarga de implementar todas las restricciones que se

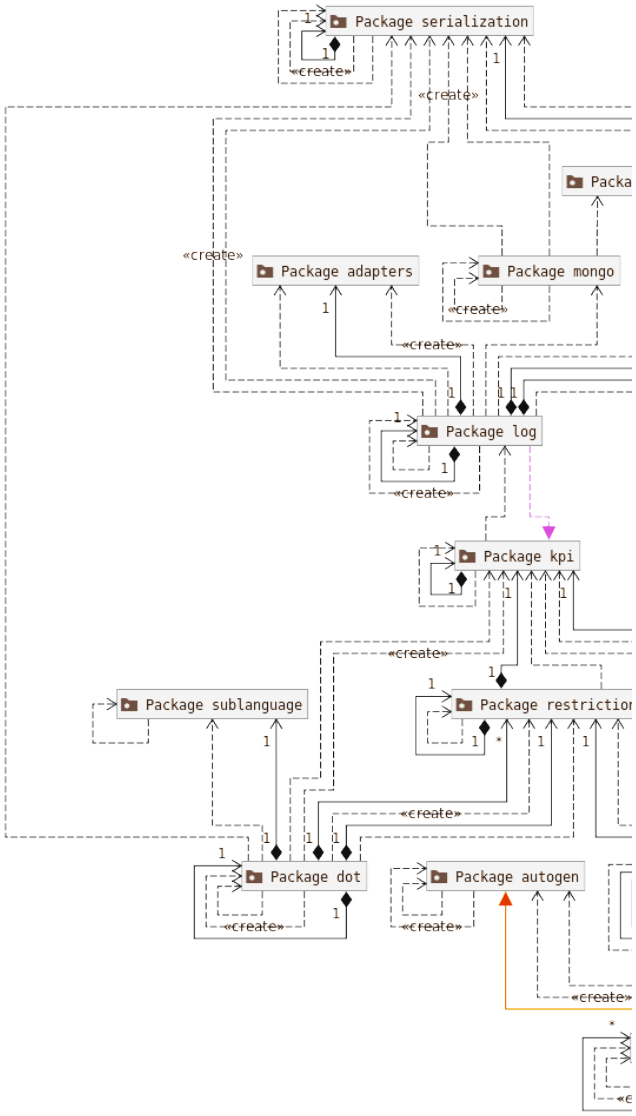


Figura 5.2: Diagrama de paquetes del primer incremento, parte 1.

pueden aplicar al algoritmo, mediante lenguajes o aplicadas directamente con código Java.

El paquete *input* controla los datos de entrada: registros, consultas y KPIs asociadas a los registros; facilitando la lectura de estos datos y preparandolos para el algoritmo.

El paquete *inverbismodel* contiene las clases de modelo de inVerbis sirve para desarrollar un adaptador de su modelo al usado por el algoritmo y facilitar el trabajo de incorporación de la herramienta a su sistema.

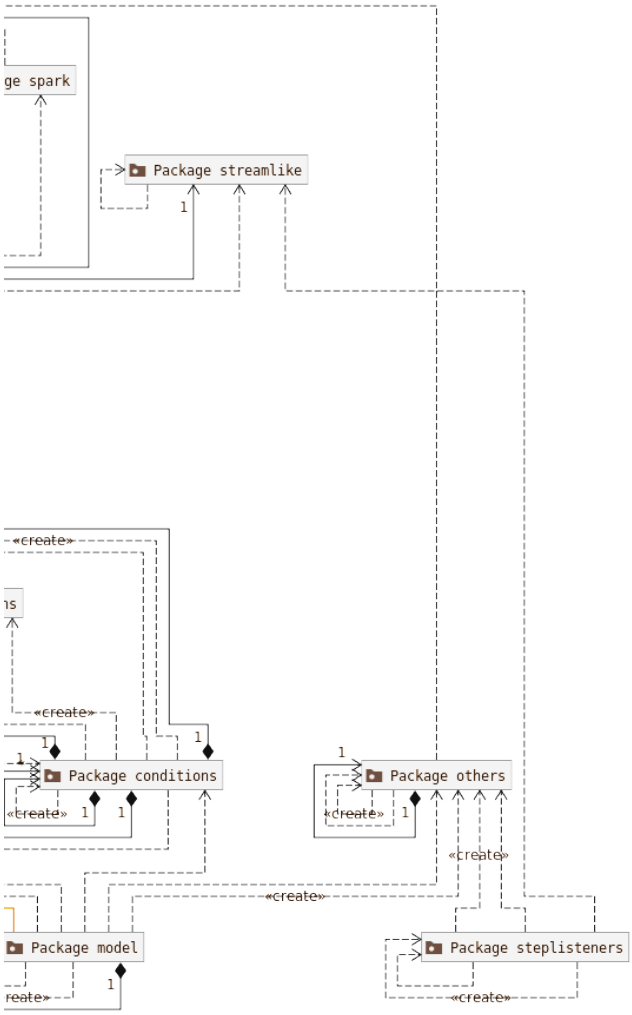


Figura 5.3: Diagrama de paquetes del primer incremento, parte 2.

El paquete *util* contiene una serie de utilidades comunes usadas por toda la aplicación por diversos motivos, por lo que a continuación incluiré los más destacables.

Entre ellas se incluyen la serialización de clases que es muy importante para Spark que debe compartir información por la red (*serialization*), abstracción de streams para tratar de la misma forma streams de java y de spark (*streamlike*), y la implementación de *listeners* del algoritmo que reciben actualizaciones a medida que se ejecuta el algoritmo, recolectan información como el tiempo medio entre tareas y son acumulables fácilmente (*steplisteners*).

5.2. Diseño e implementación

En el diseño e implementación destacan ciertas clases y su interacción, que será lo que destaque en esta sección ya que detallar todo el incremento ocuparía demasiado en la memoria, y con la visión global del apartado anterior es suficiente. Entonces se incluirán diagramas de clases y de interacción en los que solo se incluyen los elementos públicos para simplificarlos y se explican sus contenidos. En los diagramas de secuencia no se indican los condicionales por que no lo permite la herramienta, pero se describen en el texto.

5.2.1. Algoritmo

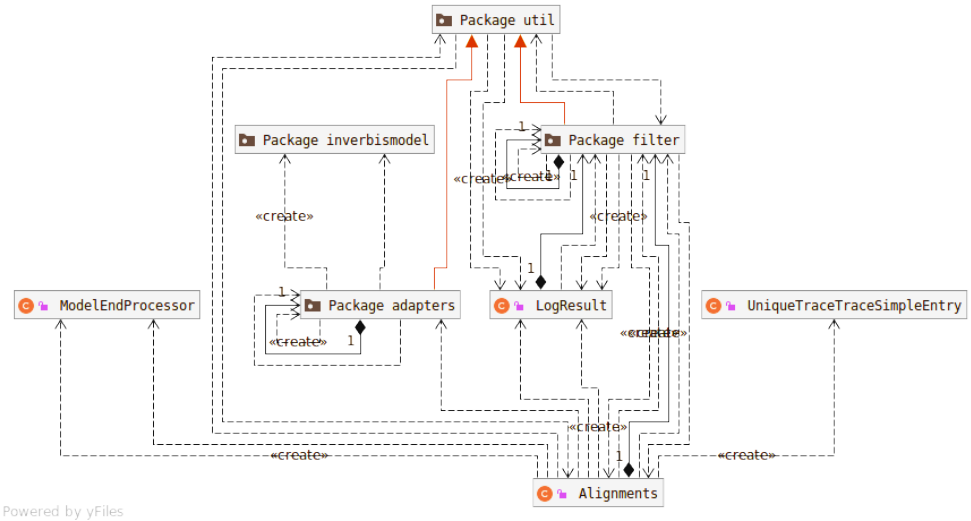


Figura 5.4: Diagrama de clases del algoritmo.

El diagrama de clases del algoritmo se encuentra en la Figura 5.4, y en ella se pueden ver las grandes clases de *LogResult*, que guarda el resultado de la ejecución del algoritmo, y *Alignments*, que realiza la ejecución del algoritmo.

En *Alignments*, si se incluyesen más datos en la imagen, se podrían observar un conjunto de constructores, un conjunto de métodos *execute*, un conjunto de métodos *executeSimple*, y unas propiedades de configuración, que son las también disponibles a través de los constructores, con valores por defecto cuando no se indican. La diferencia entre *execute* y *executeSimple* es que el primero trabaja con registros y agrega los resultados y el segundo con trazas y puede ser configurado para proporcionar más información como devolver el camino que se siguió en la traza y en la consulta marcando los saltos realizados en ambas. Como se puede ver, esta clase trabaja con el paquete adaptadores en algunos de los métodos que proporciona para facilitar el trabajo realizando también el preprocesado cuando es necesario. La configuración del algoritmo son: los *listeners*, si ejecutar en paralelo (sólo necesario si no se usa Spark, y no es recomendable

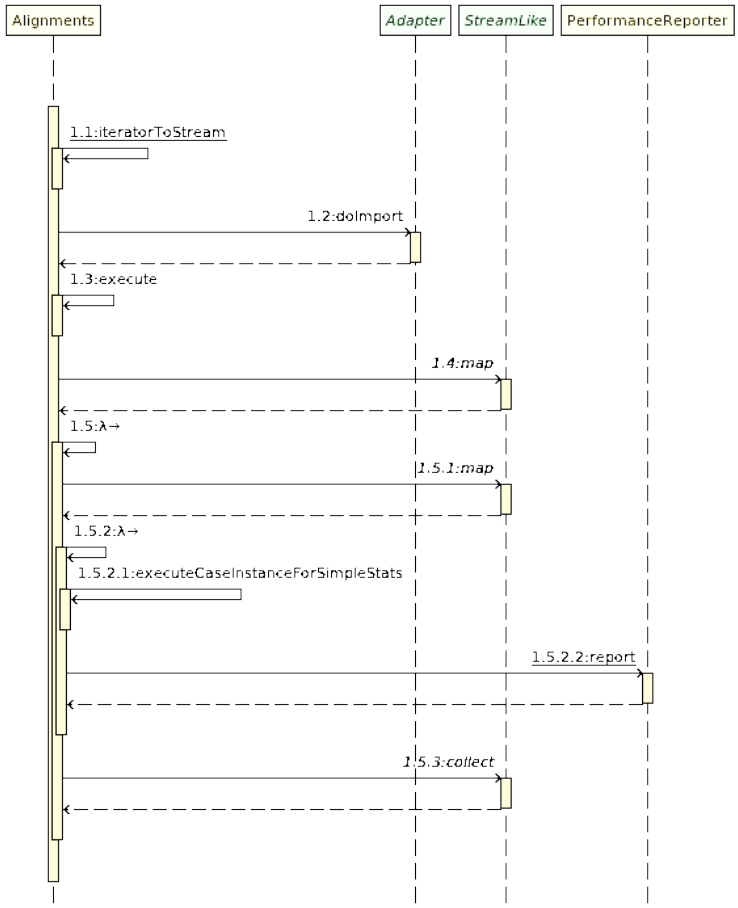


Figura 5.5: Diagrama de secuencia del algoritmo.

si se usa Spark), el fitness mínimo ya explicado, las restricciones aplicadas y el modelo de la consulta. El registro se introduce en el método *execute* de muchas formas distintas. La clase *LogResult* almacena los resultados que genera el algoritmo y ya se comentó lo que almacenaba en apartados anteriores.

En el diagrama de secuencia de la Figura 5.5 se puede ver el método `public LogResult<StepListenerType>execute(Object caseInstances, SubLogReference subLogReference) {}`, que convierte o prepara el parámetro caseInstances a algo procesable (*iteratorToStream()*, *doImport()* y/o llamada recursiva a otro método *execute()*) sólo si es necesario. Después, si se encuentra en modo de pruebas (si se ha detectado junit en el *stacktrace*, algo que se comprueba en el inicio de la máquina virtual), añade una operación de mapeo para reportar información de debug.

La segunda operación map es la que realiza la ejecución real del algoritmo sobre cada traza, que se mostrará posteriormente (como se puede ver, esta contiene otro executeCaseInstanceForSimpleStats() en su interior). Se reporta el rendimiento en el caso en el que se estén ejecutando pruebas. Por último se

agregan los resultados de todos los nodos que estén trabajando en el registro con la llamada a collect, que es la que comienza el procesado del stream ya que se trata de una operación terminal, no como los *map()* anteriores que eran operaciones intermediarias y no comienzan a procesar los datos.

Para terminar el algoritmo, se incluye el diagrama de secuencia del trabajo que realiza el algoritmo para cada traza en la Figura 5.6. Este muestra de forma general la implementación del propio algoritmo.

En este se puede ver como el algoritmo comienza creando un *MinHeap*, que servirá para ordenar los caminos que puede seguir en cada momento y escoger el de menor coste eficientemente. Se usa el cálculo del coste para podar más ramas del árbol con llamadas a *getFitness()* llegando antes al resultado. Es necesario realizar copias de las trazas porque se van a explorar varias ramas simultáneamente. A lo largo del algoritmo se ejecuta *IRestriction*, limitando también las ramas que se deben explorar del espacio de búsqueda. Por último se añaden llamadas recursivas al algoritmo para los siguientes pasos al *MinHeap*, con su coste de forma que sólo se realizarán si es necesario. El algoritmo se ejecuta mientras no se ha llegado a una solución y el *MinHeap* no está vacío.

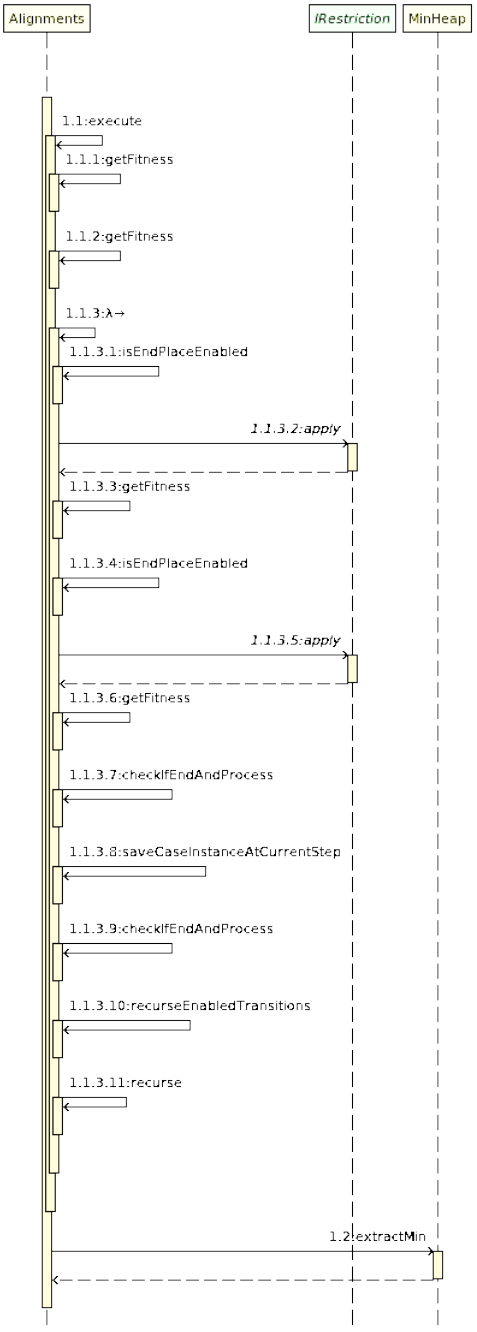


Figura 5.6: Diagrama de secuencia del algoritmo (parte 2).

5.2.2. Restricciones

La Figura 5.7 y la Figura 5.8 muestran la jerarquía de restricciones, la cual permite desarrollar clases propias de java para añadir condiciones que se deben cumplir en los pasos del algoritmo y también puede devolver un número de forma que sea el coste del paso a la hora de calcular el fitness de la traza.

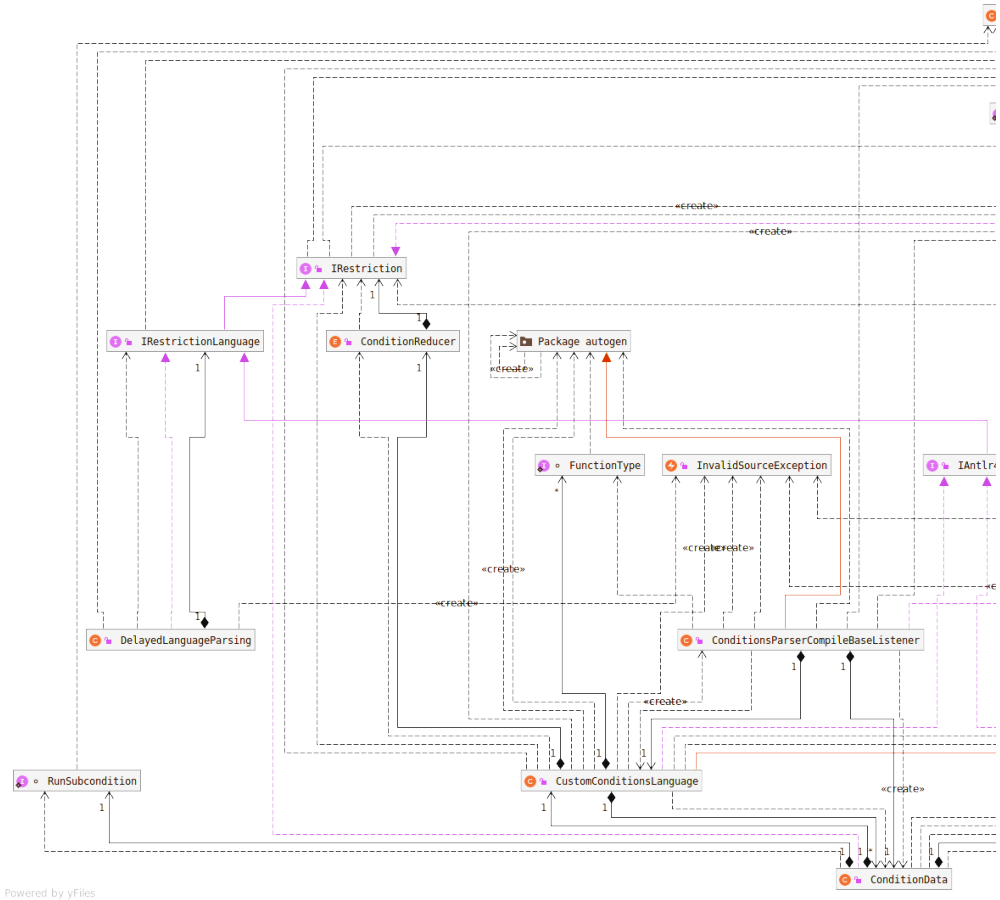


Figura 5.7: Diagrama de clases de las restricciones, parte 1.

Se proporciona una interfaz cómodas para la creación de restricciones (*IRestriction* con datos en *RestrictionData*), las cuales se combinan fácilmente entre si gracias a la que tiene métodos por defecto (de conjunción, disyunción y negación).

Las clases intermedias en el grafo sirven de utilidades para las clases inferiores, que son las que implementan los lenguajes. Se mantiene el lenguaje principal en la clase *CustomConditionsLanguage* y las dos clases inferiores, pudiendo ver en la segunda de estas todas las funciones que realiza el lenguaje. También se mantiene otro lenguaje (*DotLanguage*) basado en DOT que consiste en definir parejas actividades que cuando ocurran en ese orden se ejecutará código *JavaScript* genérico que devuelve el valor de coste del salto o *null* si es imposible continuar este camino del algoritmo. El segundo lenguaje permite muchas más posibilidades (es un lenguaje Turing completo, permitiendo a los usuarios realizar un análisis profundo en cada paso, por ejemplo, procesando datos con inteligencia artificial...), pero resulta más complejo para usuarios,

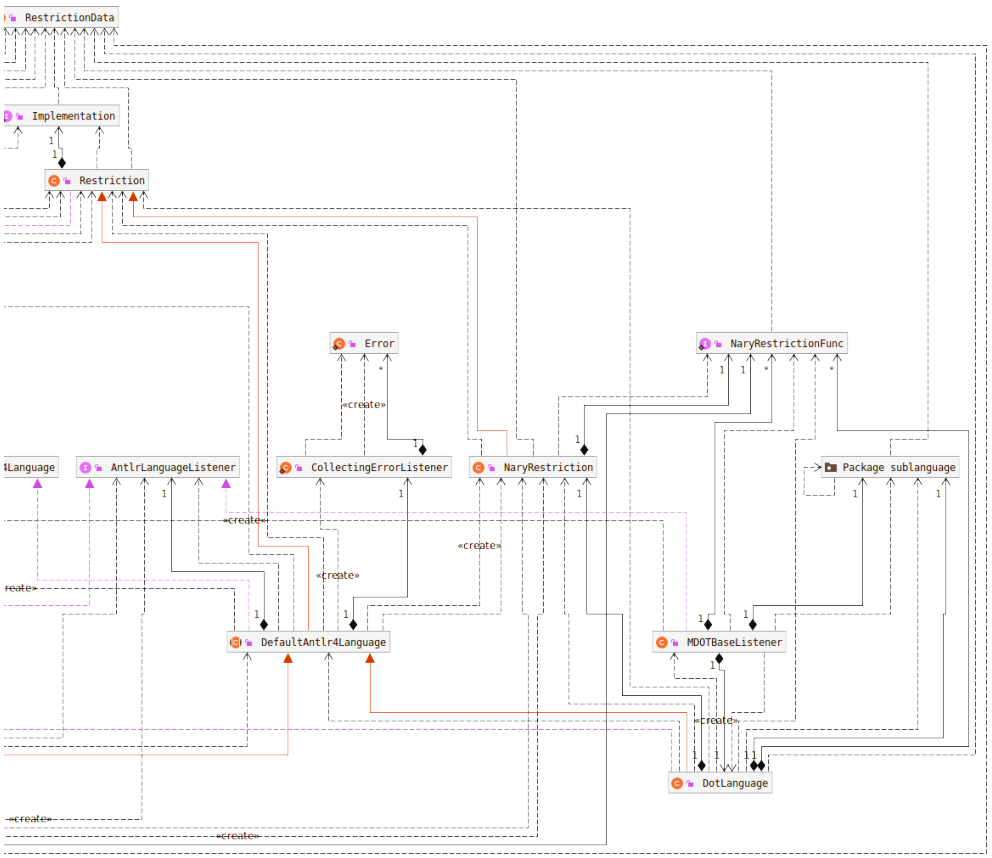


Figura 5.8: Diagrama de clases de las restricciones, parte 2.

Figura 5.9: Ejemplo de restricción usando varias características del lenguaje.

por lo que se decidió destacar el primero. Aún así, los siguientes incrementos también permiten la posibilidad de usar el segundo lenguaje.

En el gráfico se ve una clase *DelayedLanguageParsing* que retrasa el análisis sintáctico del texto del lenguaje hasta el momento de la primera ejecución (después de eso se mantiene en cache). Esto es necesario al trabajar con Spark porque no se puede enviar un lenguaje parseado por la red (por problemas de serialización), sino que para solucionarlo se parsea al llegar a cada nodo.

Como un ejemplo de lo que permitiría realizar el lenguaje de restricciones se puede ver la Figura 5.9. Para combinar los efectos de las distintas líneas

se puede escoger AND u OR, pero estos también están disponibles dentro de una línea, como se ve en la final. Este permite trabajar con números enteros o flotantes, cadenas de texto y booleanos, permitiendo la conversión entre ellos implícita (entre enteros y flotantes) y explícita mediante *casts*. Se permiten muchas operaciones, funciones y constantes. También se permite un salto mediante condición con el formato (*equation*?*ifTrue*=*expression*:*ifFalse*=*expression*) o el formato comprimido que comprueba si la entrada es *null* y si lo es la reemplaza por la segunda expresión: (*condExpression*=*expression*?:*ifFalse*=*expression*). Fue importante evitar ambigüedades en el lenguaje, de lo cual informaba la herramienta usada.

5.2.3. Adaptadores

La Figura 5.10 y la Figura 5.11 muestran la jerarquía de adaptadores, en la que destacan las 3 interfaces genéricas superiores. No se incluyen atributos ni constructores ni métodos para reducir el tamaño.

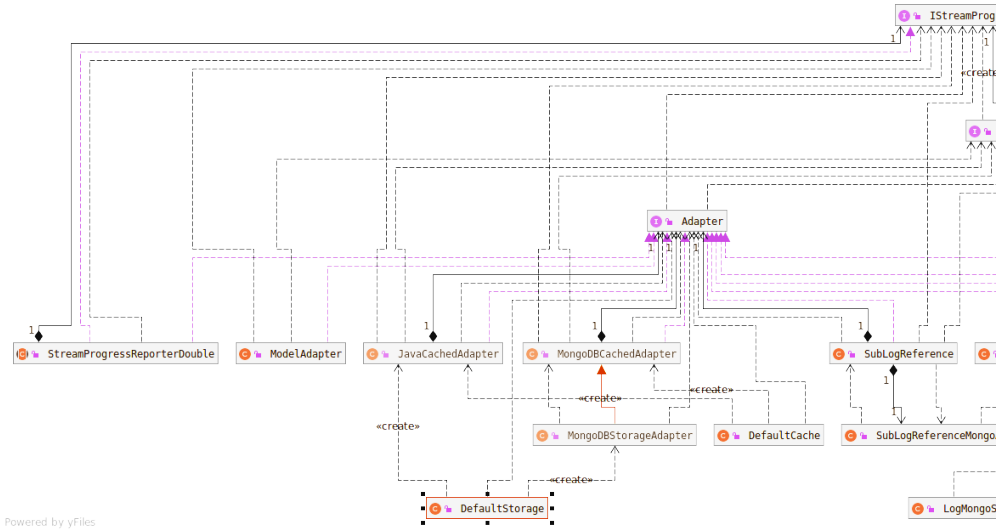


Figura 5.10: Diagrama de clases de los adaptadores, parte 1.

La interfaz *IStreamProgressReporter* es una función que cuenta el progreso a través del número de veces que fue llamada, muy facilmente conectada al procesamiento de un *Stream* de *Java* o *Spark* mediante una llamada al método *map()* del mismo. La implementación principal es *AdapterProgressReporter*, la cual filtra la información de progreso en función de un avance en el porcentaje y/o un determinado tiempo configurables, de forma que no se emitan demasiados eventos reduciendo el rendimiento.

La interfaz *Adapter* representa clases que son capaces de adaptar cualquier elemento Java (un tipo genérico) a otro, emitiendo el progreso a un *IStreamPro-*

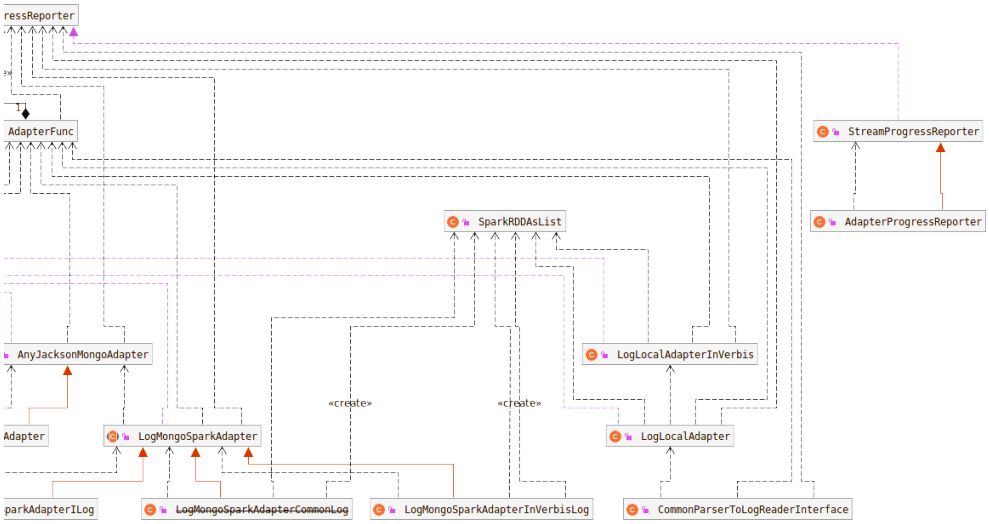


Figura 5.11: Diagrama de clases de los adaptadores, parte 2.

gressReporter. *AdapterFunc* es la interfaz funcional[2] que realiza la conversión en cada sentido. *SparkRDDAsList* es una utilidad que implementa la interfaz de una lista usando los *JavaRDD* distribuidos de *Spark*, permitiendo mantener esta optimización en modelos ajenos.

Existen muchas implementaciones de la interfaz *Adapter*, entre las que destacan la transformación del registro en el modelo de la aplicación al de inVerbis y viceversa, lo mismo para otro modelo usado en el CiTIUS y en ambos casos también la carga y exportación directa desde *MongoDB* con Spark; un adaptador entre consultas usadas por la aplicación e *inVerbis*; y un adaptador que exporta e importa referencias a registros de forma optimizada.

Además existen implementaciones de caches (tanto usando Java directamente como MongoDB) y almacenamiento persistente (usando MongoDB) aplicables fácilmente a cualquier *Adapter* programado y también son *Adapters*.

5.2.4. Input

La Figura 5.12 y la Figura 5.13 muestran el paquete *input*, formado por 3 paquetes desacoplados. Estos, en la imagen de arriba abajo, se encargan de procesar las entradas: consultas, registros y *KPI*.

El paquete de consultas contiene el lector de modelos que usa el lenguaje para generarlos. Este depende del paquete autogen, que contiene las clases autogeneradas para el análisis sintáctico. De forma previa al análisis sintáctico se procesa el lenguaje con *MyModelLanguageNoSeqForkTranspiler*. Mientras se

[2]https://docs.oracle.com/javase/8/docs/api/java/lang/FunctionalInterface.html

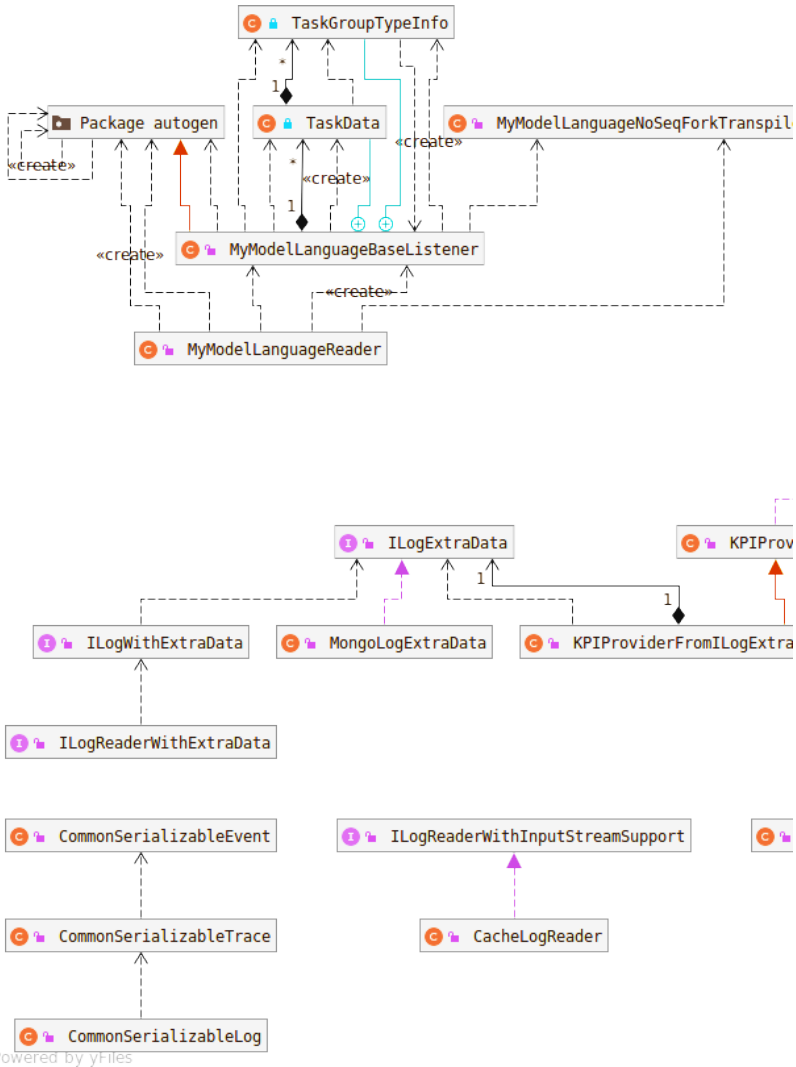


Figura 5.12: Diagrama de clases del paquete *input*, parte 1.

realiza el análisis sintáctico, la clase *MyModelLanguageBaseListener* se encarga de ir construyendo el modelo a medida que procesa *tokens*. *MyModelLanguageReader* es la implementación de la interfaz lectora de modelos para facilitar el trabajo futuro.

El paquete de *KPI* contiene algunas extensiones a la biblioteca base usada (ProDiGen) necesarias para poder manejar *KPIs* fácilmente. *ILogWithExtraData* permite añadir datos fácilmente a registros ya creados, útil para generar pruebas. *KPIProvider* es la interfaz general que facilita el acceso a los *KPIs* desde un registro de *ProDiGen*. *MLogSpark* y las clases a su derecha pertenecen, junto con las no conectadas al tercer paquete.

El paquete de registros contiene utilidades para adaptar otros modelos (las clases que comienzan por Common), facilidades para trabajar con ficheros CSV

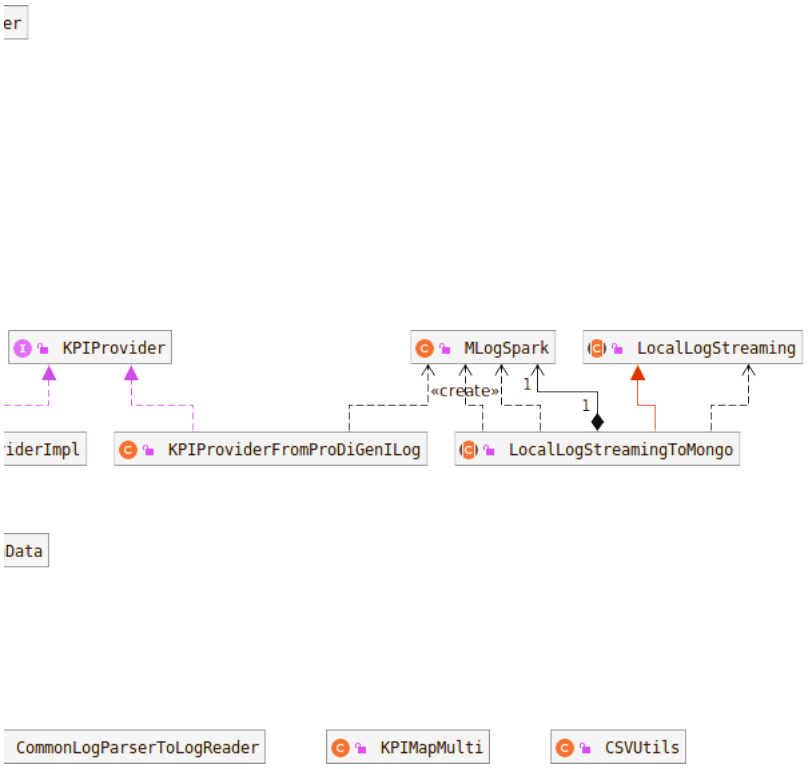


Figura 5.13: Diagrama de clases del paquete *input*, parte 2.

y dos implementaciones de la interfaz ILog de mongo: *LocalLogStreamingToMongo* que realiza la lectura de trazas al mismo tiempo que las inserta en la base de datos evitando la necesidad de mantenerlas en memoria en un sólo nodo y aumentando en gran medida el límite del registro de mayor tamaño que se puede cargar (que sólo depende de la base de datos que puede estar distribuida); y *MLogSpark* que permite preparar la lectura de registros desde *MongoDB* usando *Spark* en todo momento y es compatible con la carga que realiza la otra implementación anterior.

5.2.5. Util

La Figura 5.14 y la Figura 5.15 muestran el paquete que contiene muchas utilidades usadas en varios puntos del código. Estas están relacionadas con el manejo de registros y consultas, mongo, serialización, Spark, *listeners* del algoritmo y Streams.

La serialización es muy importante, como se puede ver en el diagrama. Tanto la clase *Jackson* como *Kryo* configuran las respectivas bibliotecas para el funcionamiento correcto con el modelo de datos usado. *Jackson* convierte las instancias a *JSON* legible mientras que *Kryo* las convierte a binario más rápido de procesar. La primera se usa, por ejemplo, para guardar registros en *MongoDB*, que permite procesarlos directamente en consultas posteriores

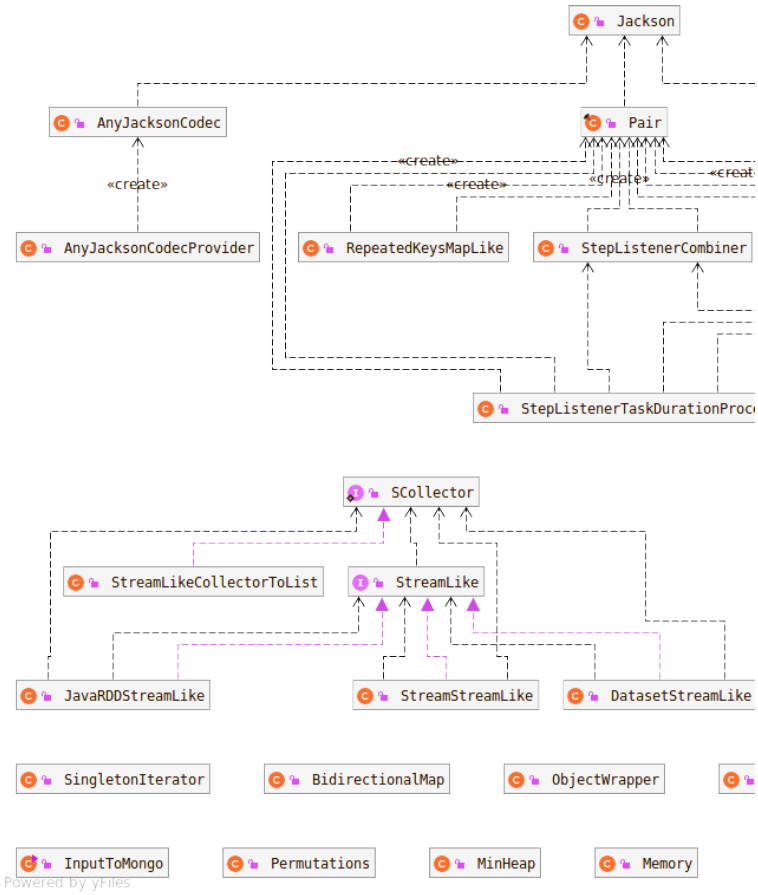


Figura 5.14: Diagrama de clases del paquete *util*, parte 1.

mejorando la localidad de los datos, mientras que la segunda se usa para la comunicación entre nodos de clúster mediante *Spark*, para mejorar la eficiencia del algoritmo. Ambos proporcionan un *Codec* para su uso en *MongoDB*.

Otro elemento destacado son los *StepListener...*, que son ampliaciones del algoritmo que calculan estadísticas y podrían realizar cualquier otra función usando los datos disponibles en cada paso del algoritmo. Actualmente se puede obtener el tiempo medio entre tareas, la duración media de cada tarea y el número de veces que se ejecuta cada transición entre tareas.

La clase *MongoUtil* almacena los métodos de acceso a la base de datos para importar y exportar registros y consultas en distintos formatos. La interfaz *StreamLike* abstrae la idea de *Streams* para permitir que sus tres implementaciones (*Streams* de *Java* y *JavaRDD* y *Dataset* de *Spark*) se traten de la misma forma en el algoritmo.

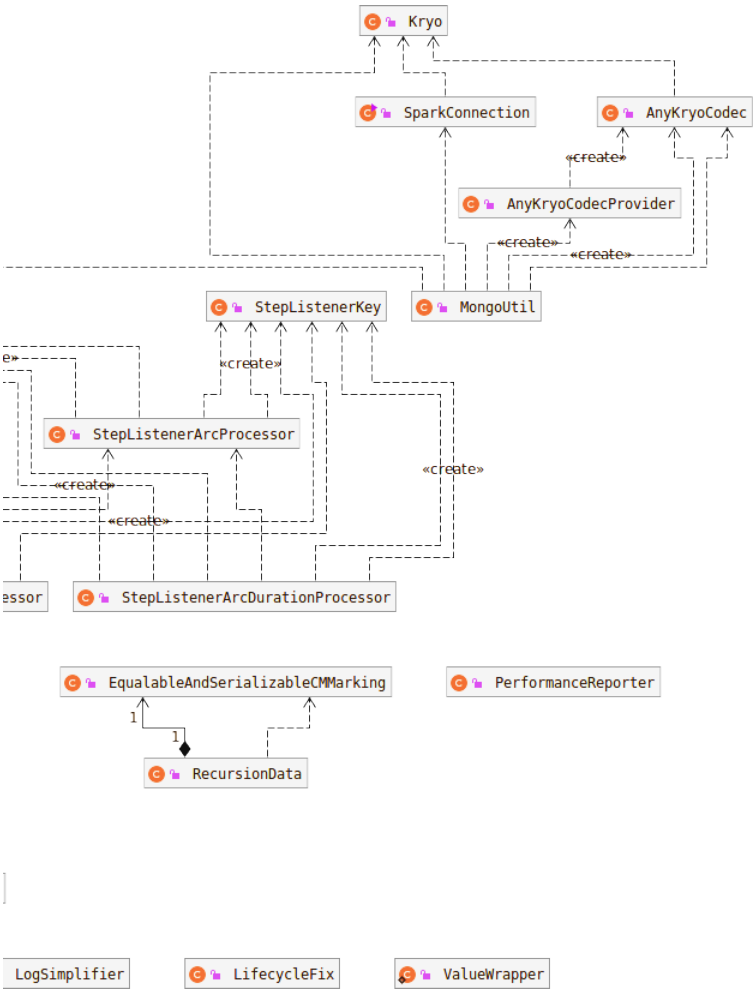


Figura 5.15: Diagrama de clases del paquete *util*, parte 2.

5.2.6. Base de datos

Se escogió una base de datos *NoSQL* ya que los registros no tienen por qué tener una estructura predeterminada (algunos eventos pueden tener más campos que otros, como una compra que guarda el conjunto de productos mientras que un inicio de sesión no los guarda). Espero que una base de datos *NoSQL* permita un acceso más eficiente, y más opciones para manejar estas estructuras.

Siempre que sea posible y útil se mantienen los datos en *JSON* legible (usando *Jackson* para convertirlos, y *MongoDB* trata eficientemente estos documentos usando el formato *BSON*), de forma que se puedan procesar fácilmente desde la propia base de datos. Por ejemplo, para almacenar modelos no se espera procesarlos desde la base de datos por las limitaciones de la misma, sino que siempre se recuperarán a la aplicación para realizar las consultas. Además en estos casos se consigue ganar eficiencia guardando los datos en binario (usando *Kryo* para la serialización).

En la Figura 5.16, la Figura 5.17, la Figura 5.18, la Figura 5.19, la Figura 5.20 y la Figura 5.21 se muestra el diseño de la base de datos, mostrando los datos que construyen y almacenan automáticamente durante las pruebas. La representación del contenido de la base de datos se realizó transformando los datos contenidos al lenguaje *GraphQL* y se representa usando la página *GraphQL Voyager*[3], obteniendo un resultado similar a un diagrama de clases.

El bloque izquierdo es el punto de entrada de la base de datos, en la que cada elemento representa las consultas que se pueden realizar sobre esa parte de la base de datos. Cada uno de estos *atributos* está unido a otra *clase* que indica lo que almacena sobre ese elemento en sus *atributos*. En este incremento la base de datos solo almacena contenidos necesarios para la ejecución de las pruebas, los cuales se añaden automáticamente por las mismas si se vacía la base de datos, como en el caso del CI.

En la segunda columna se pueden ver las colecciones almacenadas, que en este incremento coinciden con los documentos necesarios para y generados por las pruebas. Los registros almacenan su id y *KPIs* globales en colecciones de metadatos (las terminadas en Meta) mientras que las trazas se guardan en documentos separados en otra colección (1 traza por documento), con información de id, eventos de la traza, KPIs de traza y de evento, ciclo de vida, inicio y fin entre otros. Esta estructura es necesaria porque MongoDB limita el almacenamiento a 16MB por documento. Al final de esta columna se ven cargas de registros del modelo usado por inVerbis, para pruebas de adaptación entre registros.

En la tercera columna se puede ver la representación de una tarea (en la descripción de los metadatos del registro, mientras que en la secuencia de eventos se representa mediante su entero asociado para optimizar), ejemplos de KPIs globales, de traza y de evento asociadas a todos los registros usados de prueba. En la cuarta columna se indican algunos enteros largos y atributos de otras estructuras.

[3]https://apis.guru/graphql-voyager/

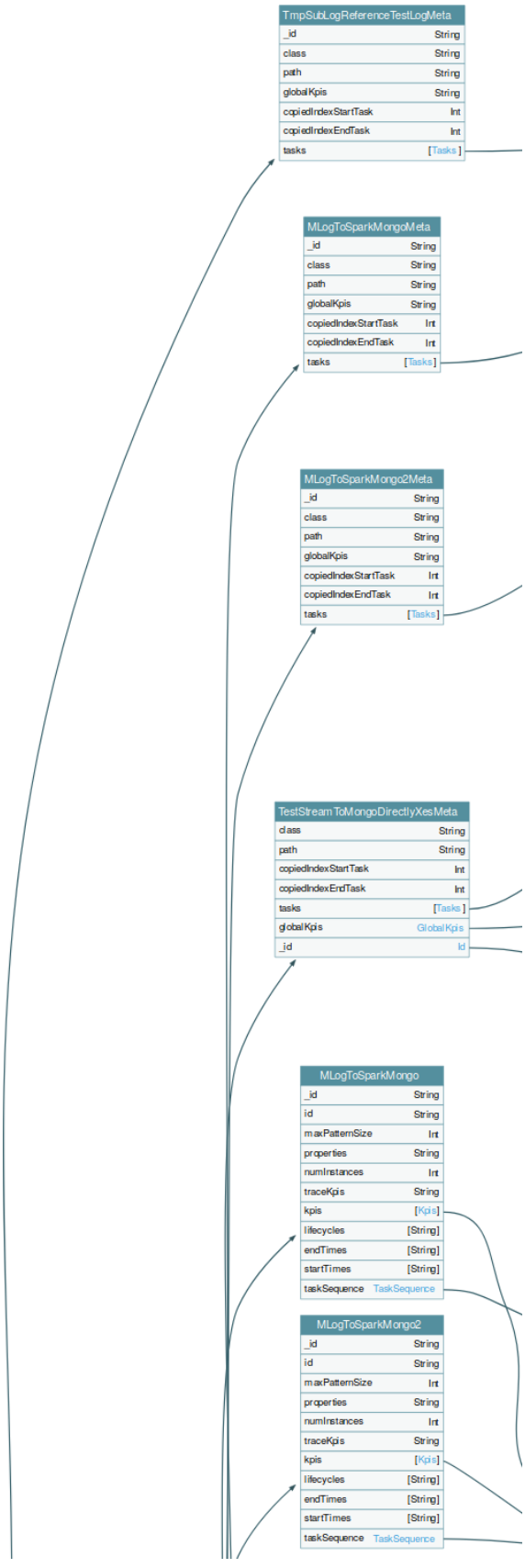


Figura 5.16: Diseño de la base de datos este incremento, parte 1.

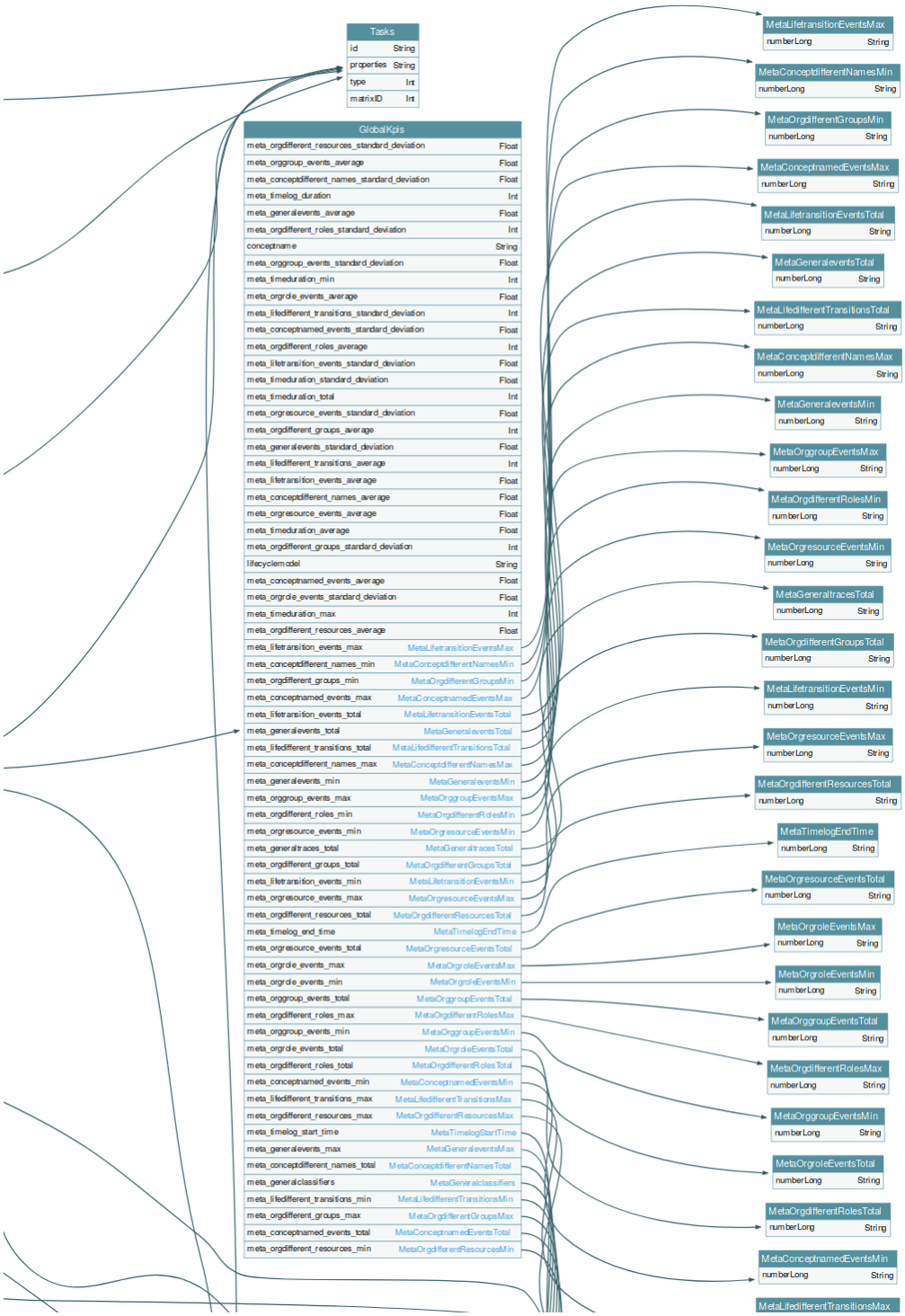


Figura 5.17: Diseño de la base de datos este incremento, parte 2.

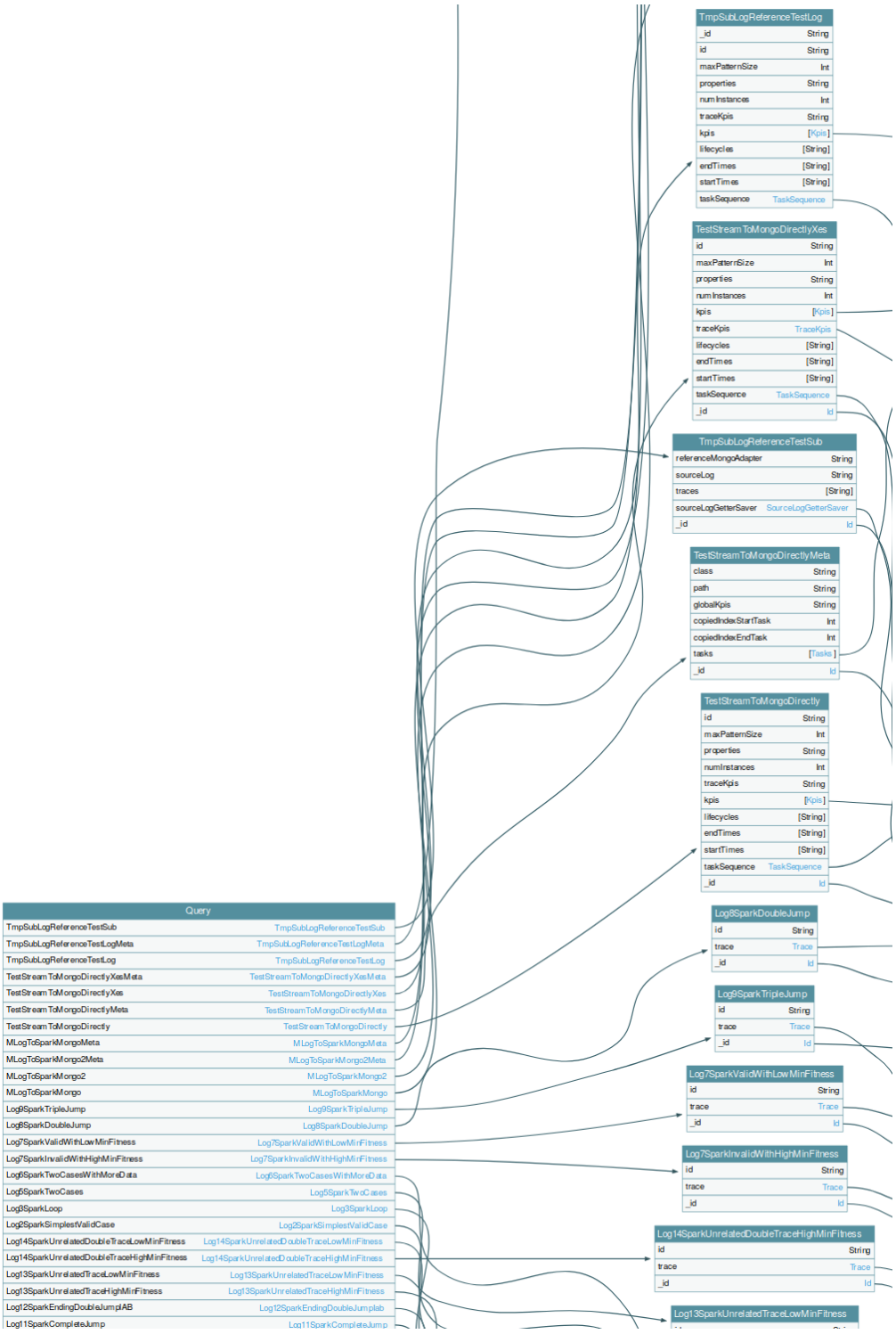


Figura 5.18: Diseño de la base de datos este incremento, parte 3.

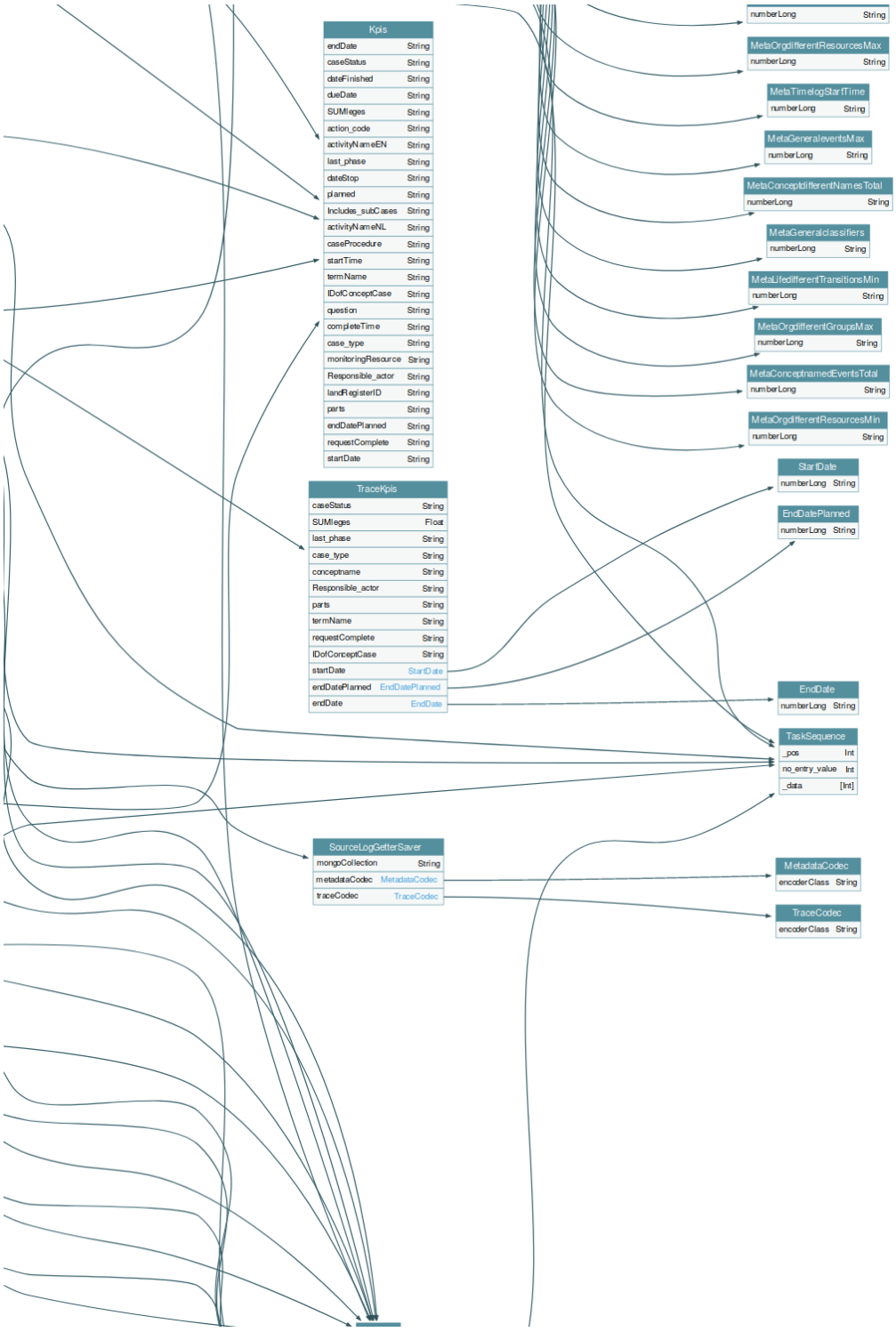


Figura 5.19: Diseño de la base de datos este incremento, parte 4.

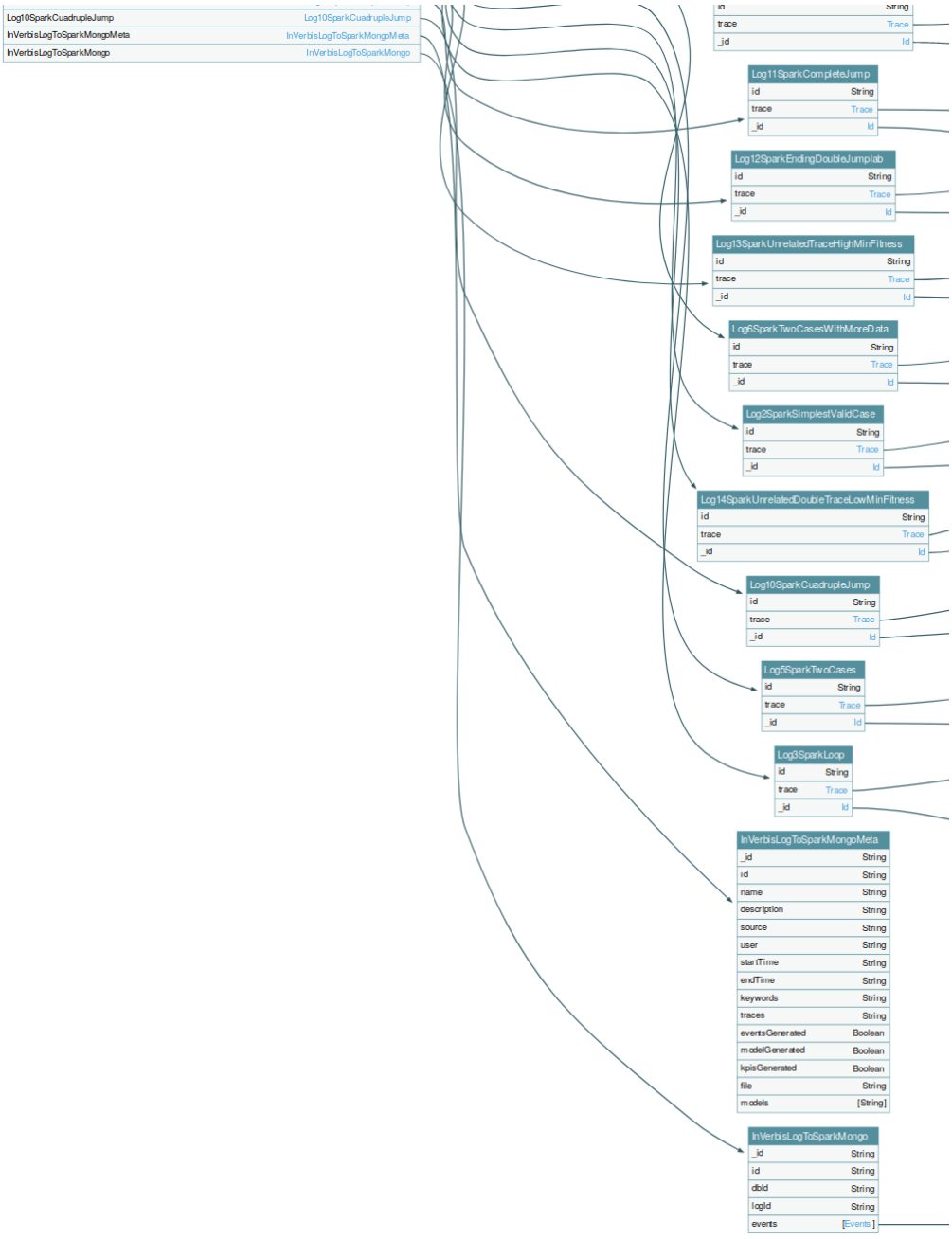


Figura 5.20: Diseño de la base de datos este incremento, parte 5.

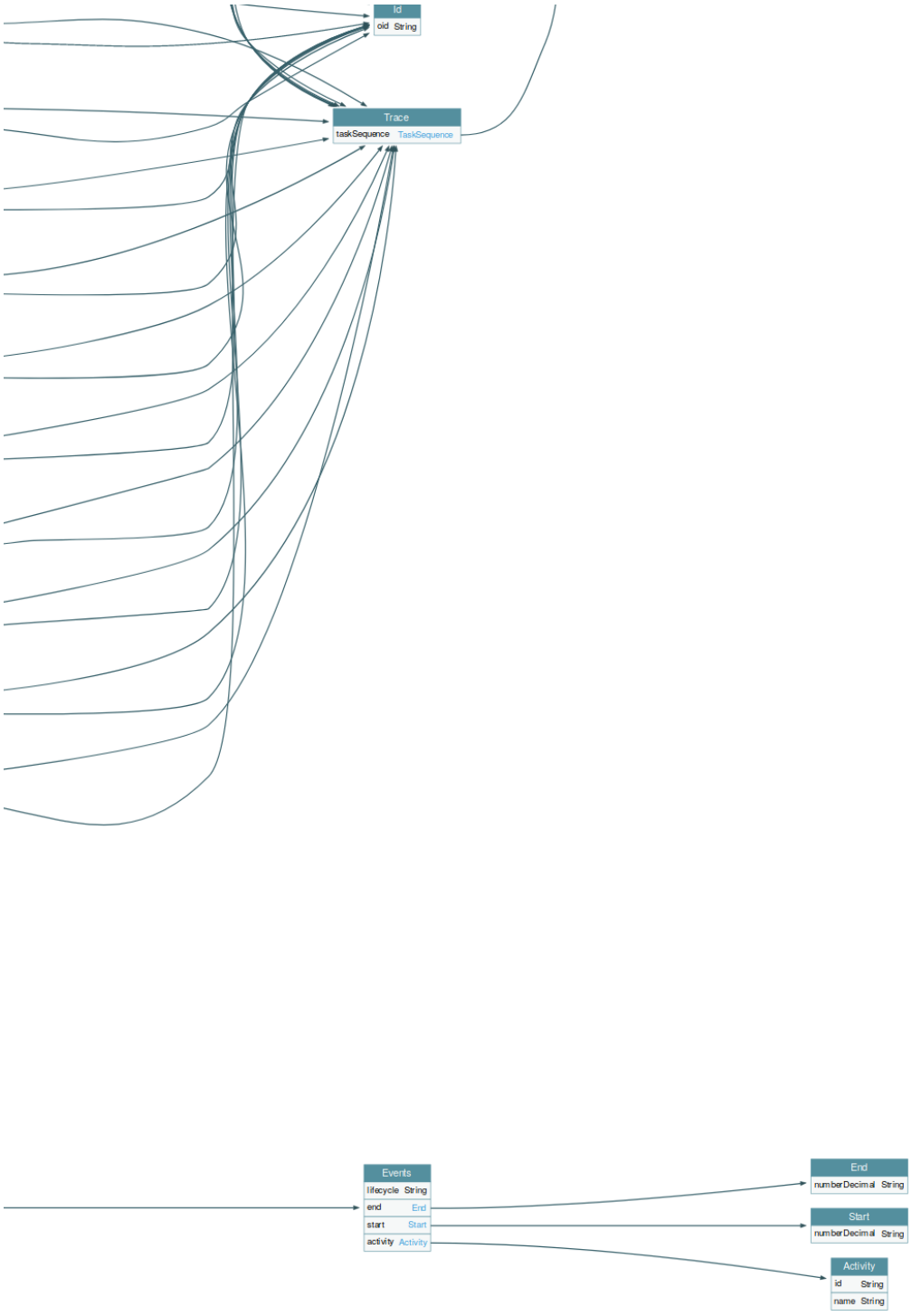


Figura 5.21: Diseño de la base de datos este incremento, parte 6.

5.3. Pruebas

5.3.1. Objetivos

Este es el módulo principal, por lo que se desarrollaron más de 300 casos de prueba que verifican que el algoritmo funciona al menos para esos casos de prueba y muchas de los otras clases también interaccionan correctamente (adaptadores, restricciones, *listeners*...). Es necesario que todas las pruebas pasen y que exista una cobertura superior al 60 % en instrucciones para considerar el trabajo de este incremento realizado, ya que es muy importante para poder realizar los siguientes incrementos. Se intentará realizar más pruebas en las partes más problemáticas porque el 80 % de los errores surgen al hacer un seguimiento del 20 % de los módulos del software (Principio de Pareto).

5.3.2. Diseño de pruebas y resultados

Este diseño de pruebas da lugar a más de 300 casos de prueba de distintas secciones del algoritmo y sus extensiones pasan correctamente.

En la Tabla 5.4 se detallan las pruebas que dieron lugar a todos los casos de prueba, que verifican los algoritmos. La técnicas siempre es el uso de JUnit 5 para ejecutar los casos de prueba en cualquier orden, y la funcionalidad abordada siempre es validar el requisito RFQ-9: Configurar y ejecutar el algoritmo salvo que se indique lo contrario, ya que es en el que se centra este incremento.

El procedimiento de ejecución de pruebas consiste en ejecutar todos los casos de prueba sin importar el orden y si alguno falla se deberá corregir el error para superar este conjunto de pruebas.

Para realizar las pruebas se iniciará el IntelliJ IDEA, y se ejecutarán los casos de pruebas JUnit 5 sobre. Esto generará un informe que incluye el método, la clase, el resultado (aceptado/fallido/saltado), un stacktrace (en los dos últimos posibles resultados) y una salida por pantalla.

Además se utilizará JaCoCo para obtener un informe que nos indique el nivel de cobertura, para saber hasta qué punto estamos probando todo el software proporcionado. En la siguiente tabla se muestran las pruebas, para cada una de las cuales se escribieron varios casos de pruebas, disponibles en Apéndice C: Incremento 1: Casos de prueba.

Nombre (ID)	StreamProgressReporterTest (PR1-1)
Objetivo	Comprobar que se reporta el progreso de las tareas que lo admitan correctamente.

Estrategia	Se comprobará que se filtran mensajes para reducir la carga cada cierto tiempo y cada cierto porcentaje configurable.
Nombre (ID)	StepListenerTaskDurationProcessorTest (PR1-2)
Objetivo	Comprobar el *listener* que obtiene la duración media de tareas.
Estrategia	Se ejecutará para varios registros, comprobando los resultados.
Nombre (ID)	TestCaseOzona, LogSimplifierTest (PR1-3)
Objetivo	Comprobar la simplificación de registros.
Estrategia	Se ejecutará para varios registros de distintos tamaños, comprobando los resultados.
Nombre (ID)	AlignmentsExample6_g24Test, AlignmentsExample3Test, AlignmentsExample5_g7SparkTest, AlignmentsExample2FullSparkTest, AlignmentsExample4SparkTest, AlignmentsExample2Test, AlignmentsExample1Test, AlignmentsExample5_g7Test, AlignmentsExample6_g24SparkTest, AlignmentsExample2SparkTest, AlignmentsExample4Test (PR1-4)
Objetivo	Comprobar la ejecución del algoritmo base y con Spark.
Estrategia	Se ejecutará para varios registros creados manualmente para mayor seguridad y registros reales, comprobando los resultados.
Nombre (ID)	CSVLoadTest (PR1-5)
Objetivo	Probará la carga de ficheros CSV con distintos parámetros.
Estrategia	Se ejecutará para varios registros, comprobando los resultados.
Nombre (ID)	StepListenerArcDurationProcessorTest, StepListenerArcProcessorTest (PR1-6)
Objetivo	Comprobar el *listener* que obtiene la duración media entre tareas.
Estrategia	Se ejecutará para varios registros, comprobando los resultados.
Nombre (ID)	AlignmentsExample5_g7_strictTest, AlignmentsExample6_g24_strictTest (PR1-7)
Objetivo	Comprobar la ejecución en modo estricto del algoritmo base y con Spark.
Estrategia	Se ejecutará para varios registros creados manualmente para mayor seguridad y registros reales, comprobando los resultados.
Nombre (ID)	ConditionsLanguageTest (PR1-8)
Objetivo	Comprobar el lenguaje de restricciones.
Estrategia	Se comprobará que una gran cantidad de lenguajes de restricciones se interpretan adecuadamente, variando las estructuras de prueba.
Nombre (ID)	TestCaseAmtegaBigPatronesSpark, TestCaseAmtegaPatrones, TestCaseAmtegaBigPatrones, TestCaseAmtega (PR1-9)
Objetivo	Ejecución del algoritmo en el registro de 1.5GB de amtega.

Estrategia	Se comprobará que el algoritmo ejecuta correctamente para procesar grandes registros.
Nombre (ID)	RestrictionTest (PR1-10)
Objetivo	Comprueba la ejecución de restricciones en código Java.
Estrategia	Se comprobará que las restricciones están correctamente integradas con el algoritmo.
Nombre (ID)	TestCaseNeerlandesesAutomaticPatterns (PR1-11)
Objetivo	Comprueba la ejecución del algoritmo con consultas descubiertas automáticamente.
Estrategia	Se comprobará que la frecuencia de las consultas descubiertas coincide con los resultados del algoritmo.
Nombre (ID)	SparkRDDAsListTest (PR1-12)
Objetivo	Comprueba la abstracción de los JavaRDD de Spark para su uso como lista.
Estrategia	Se comprobarán los métodos de la interfaz, como *forEach()* y *stream()*.
Nombre (ID)	SubLogReferenceTest (PR1-13)
Objetivo	Comprueba el manejo de referencias a subconjuntos de trazas de registros.
Estrategia	Se comprobarán la importación y exportación, además de la generación usando el algoritmo.
Nombre (ID)	ModelAdapterTest (PR1-14)
Objetivo	Comprueba el manejo de modelos.
Estrategia	Se comprobarán la importación y exportación de modelos de inVerbis.
Nombre (ID)	DotLanguageTest (PR1-15)
Objetivo	Comprobar el lenguaje de restricciones basado en DOT y JavaScript.
Estrategia	Se comprobarán la importación y exportación, además de la generación usando el algoritmo.
Nombre (ID)	LocalLogStreamingToMongoTest (PR1-16)
Objetivo	Comprueba la importación de registros directamente a MongoDB.
Estrategia	Se comprobará que a medida que se leen las trazas son insertadas en MongoDB, en distintos formatos.
Nombre (ID)	SparkLoadFromMongoTest (PR1-17)
Objetivo	Comprueba la lectura de MongoDB de registros usando Spark.
Estrategia	Se comprobará que Spark no realiza operaciones sobre MongoDB hasta la ejecución del algoritmo y se obtienen los resultados correctos.
Nombre (ID)	LogMongoSparkAdapterILogTest (PR1-18)
Objetivo	Comprueba la adaptación de registros a la interfaz compatible con ProDiGen.

Estrategia	Se comprobará que no cambian los datos contenidos en el registro y la operación se realiza sin excepciones.
Nombre (ID)	TestInputToMongo (PR1-19)
Objetivo	Comprueba la carga de registros y modelos en MongoDB.
Estrategia	Se comprobará que en MongoDB se guardan los datos siguiendo la estructura diseñada (1 documento por traza debido a la limitación de 16MB por documento de MongoDB y el guardado en formato binario en el caso de consultas).
Nombre (ID)	MyModelLanguageReaderTest (PR1-20)
Objetivo	Comprueba el lenguaje de lectura de modelos.
Estrategia	Se comprobarán muchas de las combinaciones posibles de funciones, comprobando manualmente que se generan los grafos esperados y después añadiendo comprobaciones para que no cambien.
Nombre (ID)	LogMongoSparkAdapterInVerbisLogTest (PR1-21)
Objetivo	Comprueba la lectura del modelo de inVerbis desde consultas almacenadas en MongoDB con Spark.
Estrategia	Se comprobarán los contenidos que deben ser iguales.
Nombre (ID)	TestCaseBPI2015 (PR1-22)
Objetivo	Ejecución del algoritmo en los registros obtenidos del *BPIC (Business Process Improvement Committee)*.
Estrategia	Se comprobará que el algoritmo ejecuta correctamente para procesar estos registros.
Nombre (ID)	MLogSparkTest (PR1-23)
Objetivo	Ejecución del algoritmo usando registros cargados directamente de MongoDB con Spark.
Estrategia	Se comprobará que el algoritmo ejecuta sobre logs cargados con Spark a partir del diseño de MongoDB.
Nombre (ID)	LogLocalAdapterTest (PR1-24)
Objetivo	Comprueba la recuperación de un registro completamente local compatible con ProDiGen a partir de MongoDB.
Estrategia	Se comprobará que el registro no pierde datos.
Nombre (ID)	LogMongoSparkAdapterCommonLogTest (PR1-25)
Objetivo	Comprueba la adaptación de un registro en MongoDB a un modelo común de distintos proyectos del CiTIUS.
Estrategia	Se comprobará que el registro no pierde datos en la conversión.

Tabla 5.4: Pruebas diseñadas (ver casos de prueba en el Apéndice C).

El informe de resultados de las pruebas de este incremento se muestra en la Tabla 5.5: Diseño de pruebas y resultados. Las columnas R, F, E y S significa los casos de prueba que se ejecutaron, fallaron, lanzaron errores y fueron saltados (por motivos de rendimiento) respectivamente. También se informa del tiempo de cada una de ellas, pero para analizar estos tiempos hay que tener en cuenta

que las pruebas se realizan con la información de *debug* activada hasta en las clases que realizan una gran cantidad de trabajo, por lo que los tiempos son muy superiores a los que se experienciarán en pruebas reales. Para ver resultados de tiempo más próximos a los que ocurrirán en realidad, ver la Subsección 5.3.5.

Conjunto de casos de prueba	R	F	E	S	Tiempo (s)
e.u.c.a.AlignmentsExample1Test	16	0	0	1	9.398
e.u.c.a.AlignmentsExample2SparkTest	18	0	0	2	4.965
e.u.c.a.AlignmentsExample5_g7_strictTest	6	0	0	0	339.629
e.u.c.a.CSVLoadTest	2	0	0	1	0.146
e.u.c.a.AlignmentsExample4Test	5	0	0	0	64.101
e.u.c.a.AlignmentsExample4SparkTest	4	0	0	0	84.584
e.u.c.a.AlignmentsExample3Test	4	0	0	0	0.356
e.u.c.a.TestCaseNeerlandesesAutomaticPatterns	1	0	0	1	0.001
e.u.c.a.AlignmentsExample5_g7Test	6	0	0	0	289.091
e.u.c.a.adapters.SparkRDDAsListTest	3	0	0	0	3.789
e.u.c.a.adapters.LogMongoSparkAdapterILogTest	2	0	0	0	14.462
e.u.c.a.adapters.SubLogReferenceTest	3	0	0	0	16.097
e.u.c.a.adapters.LogMongoSparkAdapterCommon	2	0	0	1	2.804
e.u.c.a.adapters.ModelAdapterTest	2	0	0	0	2.104
e.u.c.a.adapters.LogMongoSparkAdapterInVerbis	2	0	0	0	7.653
e.u.c.a.adapters.LogLocalAdapterTest	2	0	0	0	3.011
e.u.c.a.adapters.StreamProgressReporterTest	2	0	0	0	2.892
e.u.c.a.filter.language.conditions.ConditionsLangu	38	0	0	0	15.518
e.u.c.a.filter.language.dot.DotLanguageTest	8	0	0	0	1.529
e.u.c.a.filter.RestrictionTest	14	0	0	0	855.635
e.u.c.a.TestCaseAmtegaBigPatrones	7	0	0	7	0
e.u.c.a.input.model.MyModelLanguageReaderTes	41	0	0	0	1.156
e.u.c.a.input.log.MLogSparkTest	3	0	0	0	9.055
e.u.c.a.input.log.LocalLogStreamingToMongoTest	2	0	0	0	311.821
e.u.c.a.AlignmentsExample2Test	22	0	0	3	3.127
e.u.c.a.TestCaseAmtegaPatrones	7	0	0	7	0.001
e.u.c.a.AlignmentsExample6_g24Test	3	0	0	0	6.972
e.u.c.a.util.steplisteners.StepListenerTaskDuratio	2	0	0	0	10.654
e.u.c.a.util.steplisteners.StepListenerArcProcesso	1	0	0	0	874.21
e.u.c.a.util.steplisteners.StepListenerArcDuration	1	0	0	0	0.667
e.u.c.a.util.TestInputToMongo	3	0	0	3	0
e.u.c.a.util.LogSimplifierTest	12	0	0	0	0.008
e.u.c.a.TestCaseOzona	2	0	0	0	227.089
e.u.c.a.TestCaseBPI2015	15	0	0	15	0.001
e.u.c.a.TestCaseAmtega	5	0	0	4	1,592.805
e.u.c.a.AlignmentsExample5_g7SparkTest	6	0	0	0	315.608
e.u.c.a.AlignmentsExample6_g24SparkTest	3	0	0	0	6.626
e.u.c.a.AlignmentsExample6_g24_strictTest	3	0	0	0	7.487
e.u.c.a.AlignmentsExample1SparkTest	9	0	0	1	2.611
e.u.c.a.TestCaseAmtegaBigPatronesSpark	7	0	0	7	0

e.u.c.a.SparkLoadFromMongoTest	2	0	0	2	0
e.u.c.a.AlignmentsExample2FullSparkTest	17	0	0	2	8.761

Tabla 5.5: Resultados pruebas unitarias.

5.3.3. Calidad

Las herramientas de calidad permiten mejorar el código evitando errores que no se comprueban normalmente durante la compilación. En concreto se usa el analizador estático desarrollado por Facebook infer, que realiza un análisis mucho más profundo que el compilador detectando posibles *bugs*. Estos después se revisan manualmente para estudiar si son posibles o se pueden ignorar. Los resultados tras analizar todo el incremento, y corregir muchas de las sugerencias previas, fueron los siguientes.

```
Summary of the reports
  NULL_DEREFERENCE: 39
  THREAD_SAFETY_VIOLATION: 25
  INTERFACE_NOT_THREAD_SAFE: 4
```

Los problemas de seguridad entre hilos se notificaron mediante comentarios en los métodos para avisar a los usuarios que los usen, ya que resolverlas mediante sincronización de variables reduciría el rendimiento en el caso de que no se usen múltiples hilos. El rendimiento es muy importante para esta aplicación. Además, al usar Spark, la biblioteca se encarga de la sincronización entre hilos de una máquina y entre máquinas, de forma que estos avisos no se deben solucionar.

Una gran cantidad de los NULL_DEREFERENCE aparecen en clases de prueba porque se saben los contenidos que se están probando y no son *null*. Un ejemplo de los más comunes sería el siguiente.

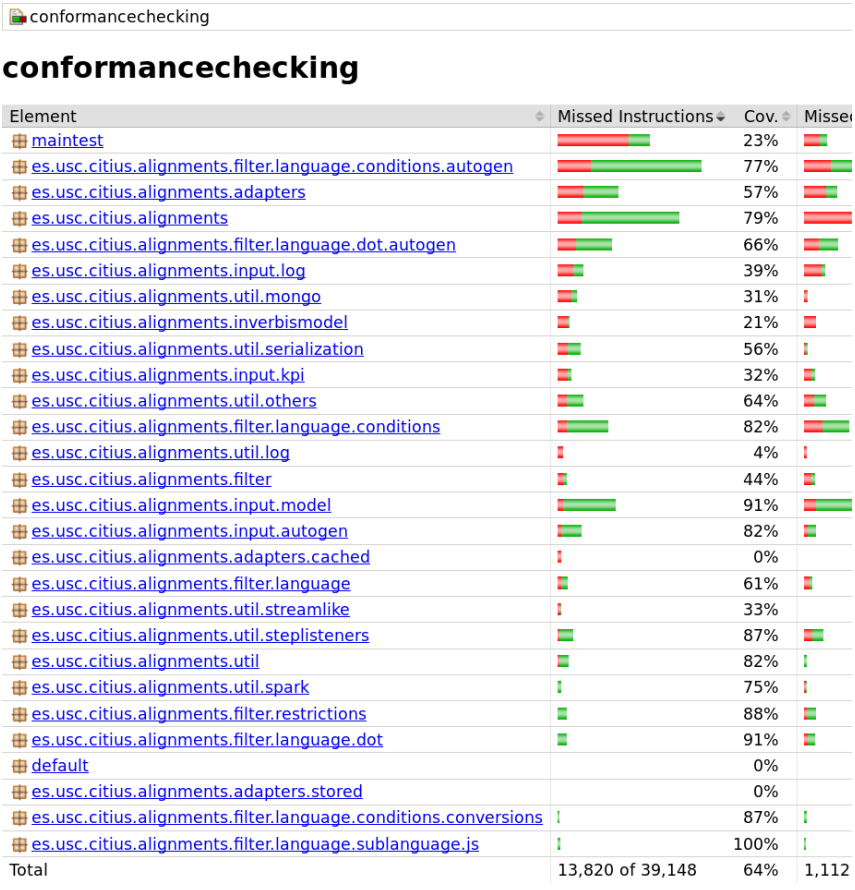
```
src/main/java/maintest/TestCaseAmtegaBigPatronesSpark.java:576: error: NULL_DEREFERENCE
object returned by `tareas.get("Start_process")` could be null and is dereferenced at l
574.           TIntHashSet subset = new TIntHashSet();
575.           set.add(subset);
576. >         tareas.get("Start_process").setInputs(set);
577.           set = new CMSet();
578.           subset = new TIntHashSet();
```

5.3.4. Cobertura

El informe de cobertura se puede ver en la Figura 5.22 y la Figura 5.23, que indica que se comprueban el **64 % (25,330 de 39,150)** de las instrucciones

de todo el código, entre muchos otros datos. Los paquetes están ordenados por instrucciones no comprobadas. El código del primer paquete (*maintest*) se usa para pruebas de rendimiento que no pueden estar en el entorno de prueba ya que este es más lento, las cuales aumentarían este porcentaje ligeramente. Estas son las clases que menor cobertura tienen porque deberían haberse considerado como pruebas. El siguiente paquete menos probado es el de clases autogeneradas para el analizador sintáctico por lo que tampoco debería considerarse en la media ya que estaría probando otro proyecto.

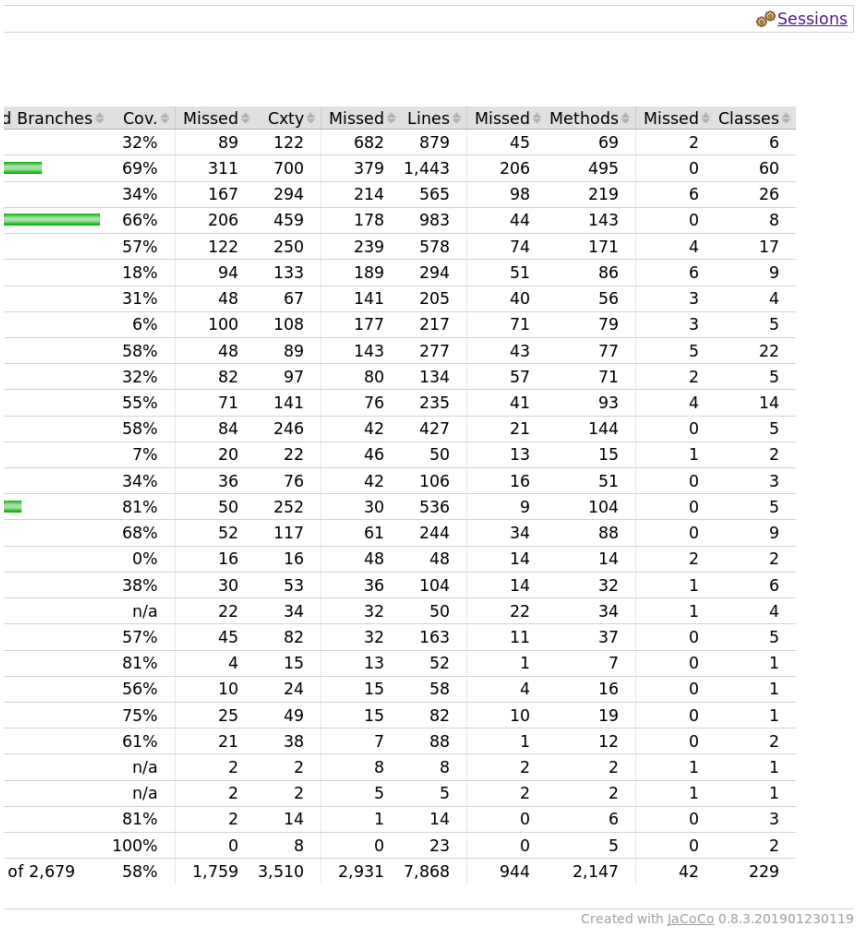
Para más información se puede descargar el artefacto generado al ejecutar todas las pruebas, el cual incluye la cobertura de los contenidos de los paquetes y también unas vistas de la cobertura de cada clase, en las cuales se muestran las líneas ejecutadas, no ejecutadas y en las que no se ejecutó algún tipo de salto. Se muestra el ejemplo de clase detallada para el algoritmo principal en la Figura 5.24, la cual es la más importante y se comprobó casi al 100 %, centrándose en los métodos principales (sin probar todos los *setters* y *getters*).


conformancechecking

conformancechecking

Element	Missed Instructions	Cov.	Misse
maintest		23%	
es.usc.citius.alignments.filter.language.conditions.autogen		77%	
es.usc.citius.alignments.adapters		57%	
es.usc.citius.alignments		79%	
es.usc.citius.alignments.filter.language.dot.autogen		66%	
es.usc.citius.alignments.input.log		39%	
es.usc.citius.alignments.util.mongo		31%	
es.usc.citius.alignments.inverbismodel		21%	
es.usc.citius.alignments.util.serialization		56%	
es.usc.citius.alignments.input.kpi		32%	
es.usc.citius.alignments.util.others		64%	
es.usc.citius.alignments.filter.language.conditions		82%	
es.usc.citius.alignments.util.log		4%	
es.usc.citius.alignments.filter		44%	
es.usc.citius.alignments.input.model		91%	
es.usc.citius.alignments.input.autogen		82%	
es.usc.citius.alignments.adapters.cached		0%	
es.usc.citius.alignments.filter.language		61%	
es.usc.citius.alignments.util.streamlike		33%	
es.usc.citius.alignments.util.steplisteners		87%	
es.usc.citius.alignments.util		82%	
es.usc.citius.alignments.util.spark		75%	
es.usc.citius.alignments.filter.restrictions		88%	
es.usc.citius.alignments.filter.language.dot		91%	
default		0%	
es.usc.citius.alignments.adapters.stored		0%	
es.usc.citius.alignments.filter.language.conditions.conversions		87%	
es.usc.citius.alignments.filter.language.sublanguage.js		100%	
Total	13,820 of 39,148	64%	1,112

Figura 5.22: Cobertura de los paquetes, parte 1.

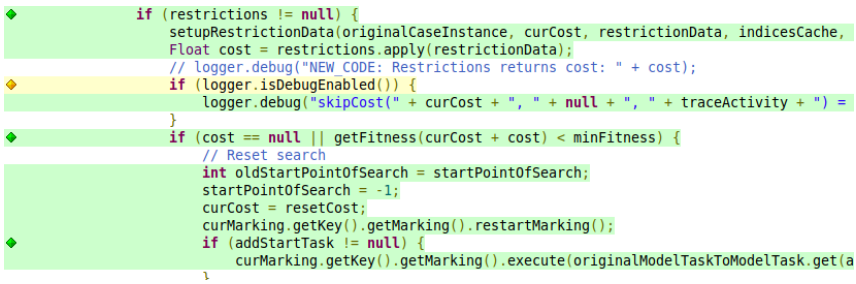


Sessions

Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
	32%	89	122	682	879	45	69	2	6
	69%	311	700	379	1,443	206	495	0	60
	34%	167	294	214	565	98	219	6	26
	66%	206	459	178	983	44	143	0	8
	57%	122	250	239	578	74	171	4	17
	18%	94	133	189	294	51	86	6	9
	31%	48	67	141	205	40	56	3	4
	6%	100	108	177	217	71	79	3	5
	58%	48	89	143	277	43	77	5	22
	32%	82	97	80	134	57	71	2	5
	55%	71	141	76	235	41	93	4	14
	58%	84	246	42	427	21	144	0	5
	7%	20	22	46	50	13	15	1	2
	34%	36	76	42	106	16	51	0	3
	81%	50	252	30	536	9	104	0	5
	68%	52	117	61	244	34	88	0	9
	0%	16	16	48	48	14	14	2	2
	38%	30	53	36	104	14	32	1	6
	n/a	22	34	32	50	22	34	1	4
	57%	45	82	32	163	11	37	0	5
	81%	4	15	13	52	1	7	0	1
	56%	10	24	15	58	4	16	0	1
	75%	25	49	15	82	10	19	0	1
	61%	21	38	7	88	1	12	0	2
	n/a	2	2	8	8	2	2	1	1
	n/a	2	2	5	5	2	2	1	1
	81%	2	14	1	14	0	6	0	3
	100%	0	8	0	23	0	5	0	2
of 2,679	58%	1,759	3,510	2,931	7,868	944	2,147	42	229

Created with JaCoCo 0.8.3.201901230119

Figura 5.23: Cobertura de los paquetes, parte 2.



```
if (restrictions != null) {
    setupRestrictionData(originalCaseInstance, curCost, restrictionData, indicesCache,
    Float cost = restrictions.apply(restrictionData);
    // logger.debug("NEW_CODE: Restrictions returns cost: " + cost);
    if (logger.isDebugEnabled()) {
        logger.debug("skipCost(" + curCost + ", " + null + ", " + traceActivity + ") =
    }
    if (cost == null || getFitness(curCost + cost) < minFitness) {
        // Reset search
        int oldStartPointOfSearch = startPointOfSearch;
        startPointOfSearch = -1;
        curCost = resetCost;
        curMarking.getKey().getMarking().restartMarking();
        if (addStartTask != null) {
            curMarking.getKey().getMarking().execute(originalModelTaskToModelTask.get(a
        }
```

Figura 5.24: Cobertura de parte del algoritmo principal.

5.3.5. Rendimiento

Se midió el rendimiento del algoritmo en varias ocasiones, e incluyo algunos resultados obtenidos en las tablas 5.6. Todos los resultados son del Log de Amtega porque es el más grande del que se dispone y no da buenos resultados

hacer medidas de rendimiento con registros pequeños (la iniciación de Spark tiene mucha más penalización).

En la primera tabla se muestran todos los parámetros y los valores que devuelve el algoritmo. Los modelos se indican con flechas, en las que → significa secuencia, ←→ significa una secuencia con un bucle entre las tareas, OR significa que se puede realizar cualquiera de las tareas y →← es un bucle de una tarea. También se muestran el número de casos y eventos de los registros. En los resultados se muestran el número de trazas aceptadas y ese mismo número si se admiten varios resultados por traza.

	Parámetros				**Resultados**	
ID	Log	Modelo buscado	# Cases	# Events	# trazas aceptadas	# aceptados
1	Amtega 1.5GB	700→004	278379	18732663	167008	613739
2	Amtega 1.5GB	700→001	278379	18732663	37421	52340
3	Amtega 1.5GB	700→621→707→004	278379	18732663	27749	31728
4	Amtega 1.5GB	(700←→071)→004	278379	18732663	164841	599728
5	Amtega 1.5GB	621→007→(004OR071)→700	278379	18732663	39570	47888
6	Amtega 1.5GB	004→700→621→707	278379	18732663	25830	29521
7	Amtega 1.5GB	Start→706←	278379	18732663	89366	89366

Tabla 5.6: Pruebas de rendimiento: información de los parámetros y resultados.

En la tabla 5.7 se pueden ver las medidas de tiempos para los parámetros anteriores. Las medidas están en milisegundos y comparan varias configuraciones. Estas se ejecutaron en el clúster *Big Data* del CiTIUS usando 14 nodos para obtener los resultados, leyendo los datos de fichero y paralelizándolos de forma previa a la medida de tiempo (se puede usar una base de datos MongoDB [8] en el propio clúster para evitar la paralelización inicial ya que cada nodo leería lo que necesita).

ID	Dataset+Spark	JavaRDD+Spark	Dataset+Spark*	JavaRDD+Spark*
1	17437	10252	20101	10957
2	16056	8880	14071	17637

Tabla 5.7: Pruebas de rendimiento: tiempos.

ID	Dataset+Spark	JavaRDD+Spark	Dataset+Spark*	JavaRDD+Spark*
3	9978	2738	10641	9017
4	11594	3700	13862	5647
5	8886	1783	7643	2000
6	9322	1548	7735	1509
7	10162	2581	10211	3171

Tabla 5.7: Pruebas de rendimiento: tiempos.

Dataset y JavaRDD son distintas tecnologías que usa Spark para manejar estructuras de datos distribuidas. Por la tabla se puede ver que JavaRDD resulta en un mayor rendimiento en todos los casos. Cuando la tabla indica Spark se probó a dejar que Spark maneje la paralelización tanto entre nodos como dentro del propio nodo, mientras que con Spark* solo se ejecuta una tarea en cada nodo y Java se encarga de paralelizar el trabajo en cada nodo usando la API de Streams [24]. Se puede ver que la versión más eficiente es JavaRDD+Spark, que es la que se usa por defecto.

En la tabla 5.8 se pueden ver otros registros con los que se realizaron medidas de rendimiento en la aplicación. Estos son registros más pequeños que tienen como objetivo medir los tiempos de iniciación de las distintas tecnologías de Spark y probar con un mayor estudio de los registros una mayor variedad de consultas sobre cada uno de ellos. La primera columna es un id que se usará después, después se indica el nombre del registro, su número de casos y eventos, el número de eventos por traza mínimo, medio y máximo, el número de actividades distintas y el número de casos tras la simplificación de los repetidos (los cuales tienen más peso al ejecutar para no dar un resultado incorrecto).

				N. Events per trace				
N.	Log	N. Cases	N. Events	min	mean	max	Act.	Simpl.
1	BPIC12-financial	13.087	288.374	5	22	177	38	4.366
2	BPIC13-clo	1.487	9.634	3	6	37	9	327
3	BPIC13-inc	7.554	80.641	3	11	125	15	2.278
4	BPIC13-op	819	3.989	3	5	24	7	819
5	BPIC15_1 (17)	1.199	54.615	4	46	103	400	1.188
6	BPIC15_2 (28)	832	46.018	3	55	134	412	828
7	BPIC15_3 (24)	1409	62499	5	44	126	385	1389

8	BPIC15_4 (21)	1.053	49.399	3	47	118	358	1.050
9	BPIC15_5 (22)	1.156	61.395	7	53	156	391	1.154
10	road-traffic	150.370	862.210	4	6	22	13	231
11	Sepsis-cases (18)	1.050	17.314	5	16	187	18	846

Tabla 5.8: Pruebas de rendimiento 2: Registros

En la tabla 5.9 se pueden ver las consultas aplicadas a los registros anteriores y los resultados obtenidos, con el tiempo necesario para obtenerlos en las distintas tecnologías ya explicadas. En la primera columna se especifica el registro usado de la tabla anterior, después la frecuencia mínima del modelo en las trazas del registro (dato proporcionado por el descubrimiento de modelos) en tanto por uno y la descripción del mismo (mediante su representación Causal Matrix), seguidos de los tiempos que llevó obtener el resultado esperado (también comprobado).

	Modelo buscado		Tiempos			
Reg.	Frec	Descripción	Dataset +Spark	JavaRDD +Spark	Dataset +Spark*	JavaRDD +Spark*
1	0.34	I(1)=[{0}]; O(1)=[{0}]; I(0)=[{1}]; O(0)=[{1}]	5.005	2.802	4.244	7.045
1	0.21	I(6)=[{5}]; O(6)=[]; I(5)=[{4}]; O(5)=[{6}]; I(4)=[{3}]; O(4)=[{5}]; I(3)=[{2}]; O(3)=[{4}]; I(2)=[{1}]; O(2)=[{3}]; I(1)=[{0}]; O(1)=[{2}]; I(0)=[]; O(0)=[{1}]	904	675	400	266
1	0.32	I(1)=[{0}]; O(1)=[{0}]; I(0)=[{1}]; O(0)=[{1}]	873	645	618	511
1	0.21	I(2)=[]; O(2)=[{0}]; I(1)=[{0}]; O(1)=[]; I(0)=[{2}]; O(0)=[{1}]	814	551	370	192
1	0.25	I(1)=[{0}]; O(1)=[]; I(0)=[]; O(0)=[{1}]	756	522	344	203
1	0.32	I(2)=[{1}]; O(2)=[]; I(1)=[{0}]; O(1)=[{2}]; I(0)=[]; O(0)=[{1}]	761	600	341	204

	Modelo buscado		Tiempos			
Reg.	Frec	Descripción	Dataset +Spark	JavaRDD +Spark	Dataset +Spark*	JavaRDD +Spark*
1	0.38	I(4)=[{1},{0}]; O(4)=[]; I(3)=[{1},{0}]; O(3)=[]; I(2)=[{1},{0}]; O(2)=[]; I(1)=[]; O(1)=[{2},{3},{4}]; I(0)=[]; O(0)=[{2},{3},{4}]	810	732	598	424
1	0.26	I(4)=[{2}]; O(4)=[{3}]; I(3)=[{4}]; O(3)=[]; I(2)=[{1}]; O(2)=[{4}]; I(1)=[{0}]; O(1)=[{2}]; I(0)=[]; O(0)=[{1}]	648	516	280	136
1	0.36	I(4)=[{3}]; O(4)=[]; I(3)=[{2}]; O(3)=[{4}]; I(2)=[{1}]; O(2)=[{3}]; I(1)=[{0}]; O(1)=[{2}]; I(0)=[]; O(0)=[{1}]	651	476	294	125
2	1.00	I(1)=[{0}]; O(1)=[]; I(0)=[]; O(0)=[{1}]	483	468	143	85
2	0.52	I(0)=[{0}]; O(0)=[{0}]	419	461	134	81
3	0.87	I(0)=[{0}]; O(0)=[{0}]	543	547	214	185
3	0.84	I(1)=[{0}]; O(1)=[]; I(0)=[]; O(0)=[{1}]	507	456	184	88
3	0.21	I(1)=[{0}]; O(1)=[{0}]; I(0)=[{1}]; O(0)=[{1}]	679	470	222	153
3	0.21	I(4)=[{3}]; O(4)=[]; I(3)=[{2}]; O(3)=[{4}]; I(2)=[{0}]; O(2)=[{3}]; I(1)=[]; O(1)=[{0}]; I(0)=[{1}]; O(0)=[{2}]	582	486	190	117
3	0.30	I(1)=[{0}]; O(1)=[]; I(0)=[]; O(0)=[{1}]	726	418	191	102
3	0.25	I(2)=[{0}]; O(2)=[{1}]; I(1)=[{2}]; O(1)=[]; I(0)=[]; O(0)=[{2}]	545	566	147	87
3	0.21	I(2)=[{0}]; O(2)=[]; I(1)=[]; O(1)=[{0}]; I(0)=[{1}]; O(0)=[{2}]	681	443	158	89
3	0.29	I(1)=[{0}]; O(1)=[{0}]; I(0)=[{1}]; O(0)=[{1}]	666	566	201	178
4	0.30	I(0)=[{0}]; O(0)=[{0}]	4.098	582	99	78
5	0.41	I(1)=[{0}]; O(1)=[]; I(0)=[]; O(0)=[{1}]	532	635	147	124

	Modelo buscado		Tiempos			
Reg.	Frec	Descripción	Dataset +Spark	JavaRDD +Spark	Dataset +Spark*	JavaRDD +Spark*
5	0.24	I(7)=[{0}]; O(7)=[]; I(6)=[{5}]; O(6)=[]; I(5)=[{4}]; O(5)=[{6}]; I(4)=[{3}]; O(4)=[{5}]; I(3)=[{2}]; O(3)=[{4}]; I(2)=[{0}]; O(2)=[{3}]; I(1)=[{0}]; O(1)=[]; I(0)=[]; O(0)=[{2},{7},{1}]	5.160	458	121	165
5	0.20	I(2)=[{1}]; O(2)=[{0}]; I(1)=[]; O(1)=[{2}]; I(0)=[{2}]; O(0)=[]	533	480	116	80
5	0.25	I(3)=[]; O(3)=[{2}]; I(2)=[{1},{3},{0}]; O(2)=[]; I(1)=[]; O(1)=[{2}]; I(0)=[]; O(0)=[{2}]	512	499	121	172
5	0.22	I(2)=[{1}]; O(2)=[{0}]; I(1)=[]; O(1)=[{2}]; I(0)=[{2}]; O(0)=[]	486	435	119	76
6	0.22	I(2)=[{0}]; O(2)=[]; I(1)=[]; O(1)=[{0}]; I(0)=[{1}]; O(0)=[{2}]	731	465	122	85
6	0.23	I(1)=[]; O(1)=[{0}]; I(0)=[{1}]; O(0)=[]	639	390	104	71
6	0.25	I(1)=[{0}]; O(1)=[]; I(0)=[]; O(0)=[{1}]	645	384	109	79
6	0.20	I(2)=[{1}]; O(2)=[]; I(1)=[{0}]; O(1)=[{2}]; I(0)=[]; O(0)=[{1}]	533	444	107	98
6	0.24	I(1)=[{0}]; O(1)=[]; I(0)=[]; O(0)=[{1}]	488	474	123	68
7	0.22	I(2)=[{0}]; O(2)=[]; I(1)=[]; O(1)=[{0}]; I(0)=[{1}]; O(0)=[{2}]	646	396	134	146
7	0.21	I(4)=[{0}]; O(4)=[{1}]; I(3)=[{2}]; O(3)=[]; I(2)=[{1}]; O(2)=[{3}]; I(1)=[{4}]; O(1)=[{2}]; I(0)=[]; O(0)=[{4}]	537	341	120	66
7	0.22	I(1)=[{0}]; O(1)=[]; I(0)=[]; O(0)=[{1}]	582	420	113	55

	Modelo buscado		Tiempos			
Reg.	Frec	Descripción	Dataset +Spark	JavaRDD +Spark	Dataset +Spark*	JavaRDD +Spark*
7	0.21	I(3)=[{1}]; O(3)=[{2}]; I(2)=[{3}]; O(2)=[]; I(1)=[{0}]; O(1)=[{3}]; I(0)=[]; O(0)=[{1}]	586	491	118	94
7	0.32	I(2)=[{1}]; O(2)=[{0}]; I(1)=[]; O(1)=[{2}]; I(0)=[{2}]; O(0)=[]	539	426	115	63
8	0.21	I(1)=[]; O(1)=[{0}]; I(0)=[{1}]; O(0)=[]	1.151	491	128	67
8	0.27	I(1)=[{0}]; O(1)=[]; I(0)=[]; O(0)=[{1}]	1.354	421	113	67
8	0.28	I(2)=[{0}]; O(2)=[{1}]; I(1)=[{2}]; O(1)=[]; I(0)=[]; O(0)=[{2}]	1.359	413	138	205
8	0.25	I(6)=[{5}]; O(6)=[{4}]; I(5)=[{3}]; O(5)=[{6}]; I(4)=[{6}]; O(4)=[]; I(3)=[{2}]; O(3)=[{5}]; I(2)=[{1}]; O(2)=[{3}]; I(1)=[{0}]; O(1)=[{2}]; I(0)=[]; O(0)=[{1}]	1.261	439	123	69
8	0.30	I(1)=[{0}]; O(1)=[]; I(0)=[]; O(0)=[{1}]	1.274	467	105	90
9	0.21	I(2)=[]; O(2)=[{0}]; I(1)=[{0}]; O(1)=[]; I(0)=[{2}]; O(0)=[{1}]	527	416	128	75
9	0.26	I(2)=[{1}]; O(2)=[]; I(1)=[{0}]; O(1)=[{2}]; I(0)=[]; O(0)=[{1}]	519	411	130	51
9	0.24	I(6)=[{5}]; O(6)=[]; I(5)=[{4}]; O(5)=[{6}]; I(4)=[{3}]; O(4)=[{5}]; I(3)=[{2}]; O(3)=[{4}]; I(2)=[{1}]; O(2)=[{3}]; I(1)=[{0}]; O(1)=[{2}]; I(0)=[]; O(0)=[{1}]	528	577	117	59
9	0.24	I(1)=[]; O(1)=[{0}]; I(0)=[{1}]; O(0)=[]	480	440	86	51
9	0.26	I(1)=[{0}]; O(1)=[]; I(0)=[]; O(0)=[{1}]	607	443	103	52
10	0.31	I(3)=[{1}]; O(3)=[{2}]; I(2)=[{3}]; O(2)=[]; I(1)=[{0}]; O(1)=[{3}]; I(0)=[]; O(0)=[{1}]	308	363	96	52

Modelo buscado			Tiempos			
Reg.	Frec	Descripción	Dataset +Spark	JavaRDD +Spark	Dataset +Spark*	JavaRDD +Spark*
10	0.69	I(2)=[{1}]; O(2)=[]; I(1)=[{0}]; O(1)=[{2}]; I(0)=[]; O(0)=[{1}]	349	349	79	48
10	0.31	I(3)=[{1}]; O(3)=[{2}]; I(2)=[{3}]; O(2)=[]; I(1)=[{0}]; O(1)=[{3}]; I(0)=[]; O(0)=[{1}]	5.401	987	3.774	807
10	0.69	I(2)=[{1}]; O(2)=[]; I(1)=[{0}]; O(1)=[{2}]; I(0)=[]; O(0)=[{1}]	5.344	618	3.528	619
11	0.28	I(1)=[]; O(1)=[{0}]; I(0)=[{1}]; O(0)=[]	453	581	86	61
11	0.29	I(2)=[{1}]; O(2)=[]; I(1)=[{0}]; O(1)=[{2}]; I(0)=[]; O(0)=[{1}]	419	429	80	46
11	0.27	I(1)=[{0}]; O(1)=[]; I(0)=[]; O(0)=[{1}]	496	376	72	97
11	0.20	I(0)=[{0}]; O(0)=[{0}]	475	508	96	76
11	0.29	I(5)=[{1},{4}]; O(5)=[]; I(4)=[]; O(4)=[{5}]; I(3)=[{2}]; O(3)=[]; I(2)=[{0}]; O(2)=[{3}]; I(1)=[{0}]; O(1)=[{5}]; I(0)=[]; O(0)=[{1},{2}]	530	495	135	94

Tabla 5.9: Pruebas de rendimiento 2: Consultas y resultados.

La primera fila tiene mayores tiempos porque se cuenta la inicialización de Spark que no es necesaria para las siguientes ejecuciones. Al usar registros de menor tamaño, se pudieron realizar pruebas más exhaustivas, obtener estadísticas sobre los registros y descubrir modelos que usar como consultas automáticamente, con una determinada frecuencia mínima. Las diferencias entre las tecnologías son más difíciles de apreciar en estos registros (siendo JavaRDD+Spark* mejor en general, probablemente porque resulta más lento inicializar Spark a usar la paralelización de Java para tratar registros pequeños), pero se consiguieron probar muchas más consultas comprobando que los resultados sean los mismos que los obtenidos por el programa que descubre procesos y se determinó una relación clara entre la longitud de las trazas (número de eventos que contienen) y el rendimiento del algoritmo.

5.3.6. Informe ejecutivo

En este incremento se superaron todas las pruebas consiguiendo una cobertura media de 64 %, la cual se centra en los puntos centrales como el propio algoritmo y los lenguajes y permite dar cierta seguridad sobre el correcto funcionamiento del incremento; superando el límite impuesto en los objetivos.

También se comprobó que la calidad mediante un analizador estático de código y el rendimiento de la herramienta es el adecuado al comparar distintas tecnologías y escoger la óptima para registros de mayor tamaño.

Capítulo 6

Incremento 2: servidor

El objetivo de este incremento es desarrollar un servidor web que ofrezca una *API* de servicios *REST* para que los desarrolladores puedan utilizar este algoritmo de forma remota y se pueda desarrollar una interfaz usándolo.

6.1. Arquitectura

6.1.1. Descripción

El servidor web se implementará con *Spring Boot* [20], permitiendo autenticación y asociándolos procesos y registros subidos a cada usuario, no permitiéndole acceder a los datos privados de otros usuarios; pero se podría aumentar permitiendo compartir algunos modelos de forma pública, para reducir el uso de disco.

El servidor debe permitir la ejecución del algoritmo en el propio servidor para permitir una configuración y prueba sencillas, y también el uso de un clúster externo en el que se pueda ejecutar Spark para acelerar las consultas.

Además, como se espera que algunas de las tareas sean de un tiempo de ejecución elevado, el servidor informará del progreso de algunas consultas a través de *Server Sent Events* (lo que supone sumar el progreso asíncrono de los trabajadores implicados). Los eventos enviados por el servidor permiten que el usuario no deba mantener la conexión de la consulta abierta (lo cual podría suponer un problema si las consultas pueden durar horas o días), pero se mantiene informando del progreso de la misma en conexiones posteriores. Además, esto le permite, por ejemplo, realizar varias ejecuciones del algoritmo de forma simultánea viendo el progreso de ambas y pudiendo cerrar sesión y desconectar completamente el cliente para después volver a iniciar sesión y comprobar los resultados obtenidos.

Los registros y modelos se podrán cargar en distintos formatos para mejorar la usabilidad, y también exportar en distintos formatos. Por ejemplo, se permitirá cargar registros en CSV (por ser muy comunes en aplicaciones simples), incluyendo la selección de columnas (caso, evento, fecha y el formato de la fecha

la cual dispone de una ayuda leyendo el fichero) en la interfaz gráfica, con el procesado apropiado en el servidor.

El servidor ofrecerá una descripción *OpenAPI* de los servicios de los que dispone, de forma que se pueden desarrollar nuevos clientes (y autogenerarlos para acelerar el desarrollo) o permitir que los usuarios automaticen ciertas tareas fácilmente actuando directamente sobre esta API sin tener que acceder a la interfaz.

Los motivos por los que se escogió cambiar a *Gradle* en lugar de *Maven* para el desarrollo de este proyecto fueron:

Flexibilidad. La herramienta es muy extensible y fácil de integrar con otras creando tareas nuevas, como se ve en los proyectos.

Rendimiento. La compilación incremental, las caches de compilación y el demonio de compilación aceleran las tareas más de lo que tardaría con maven.

Manejo de dependencias. Permite substituir dependencias indirectas[1] del proyecto más fácilmente lo que facilitó, por ejemplo, substituir las dependencias obsoletas de log4j de proyectos que no podía modificar por logback y obtener un mejor sistema de log integrado con Spring.

Integraciones. Resulta más fácil integrar las herramientas usadas en el proyecto.

La interfaz genera muchos archivos pequeños (*chunks*) para evitar que el usuario tenga que descargar todo el código de la aplicación cuando sólo va a usar ciertas vistas de la misma y para mejorar el funcionamiento de las caches del navegador en cambios de una vista. Esto significa que existen bastantes ficheros pequeños que se descargarán en paralelo.

Para esta tarea, HTTP/2 optimiza el tiempo de descarga con respecto a HTTP/1.1 que es usado por Spring por defecto. Para poder usar HTTP/2 se necesita usar HTTPS, por lo que se creo una clave de prueba para desplegar con HTTPS el servidor y el clúster. Java sólo permite usar HTTP/2 con al menos Java 9. Spark no soporta Java 9 y superiores aunque la aplicación compila y ejecuta correctamente hasta que se usa Spark. Entonces se usa Java 8 y habría que esperar a una actualización de Spark para que se solucione el problema de compatibilidad. El clúster está protegido con un formulario de login al igual que el servidor.

Se crearon 3 subproyectos para conseguir este objetivo ya que se debe proporcionar un servidor y un ejecutable para el clúster. Esto permite compartir el código del modelo que es común a ambos proyectos y evitar usar todas las dependencias transitivas del servidor si se hiciera que el clúster dependiera de él. Para tener una idea más clara de la estructura de estos proyectos, se muestra

[1] Dependencias de otros proyectos usados como bibliotecas

la tarea de compilación (./gradlew install) configurada en Gradle en la Figura 6.1 y la Figura 6.2, en la que *bootServerJar* y *bootClusterJar* son las tareas que generan el jar ejecutable para el servidor y el cluster respectivamente.

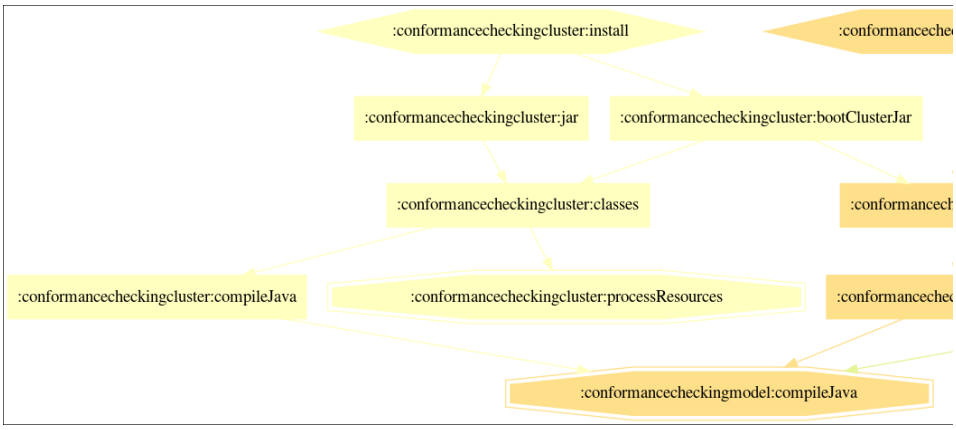


Figura 6.1: Tareas gradle de los 3 subproyectos, parte 1.

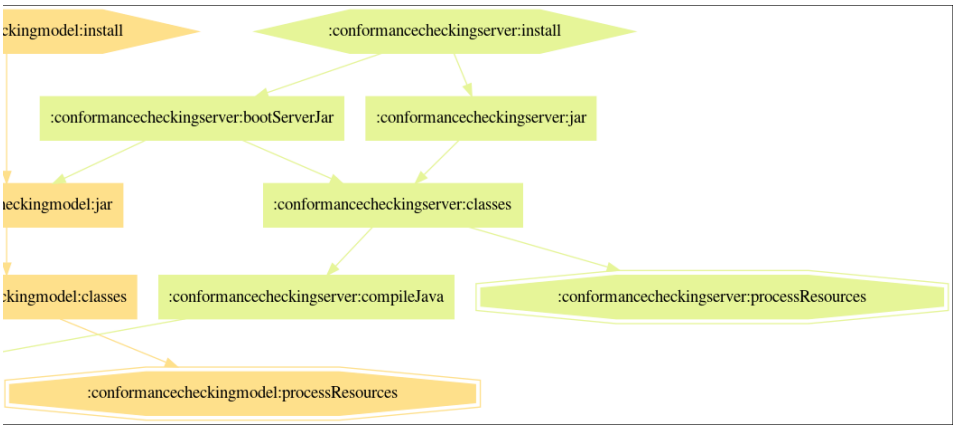


Figura 6.2: Tareas gradle de los 3 subproyectos, parte 2.

6.1.2. Diagrama y explicación de paquetes

La estructura de los paquetes en los tres proyectos es la recomendada en Spring Boot: *config* guarda las clases de configuración, *filter* contiene los filtros, que en este caso son de autenticación y autorización de usuarios, *controller* ofrece los servicios, *model* contiene las clases del modelo, *repository* contiene las clases de acceso a repositorios, dividas en normales y acceso reactivo (explicado después) (. . .r.normal y . . .r.reactive), y *service* contiene los servicios de Spring ofrecidos, como el de email y el del algoritmo.

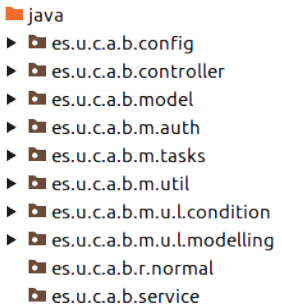


Figura 6.3: Estructura de paquetes: modelo común.

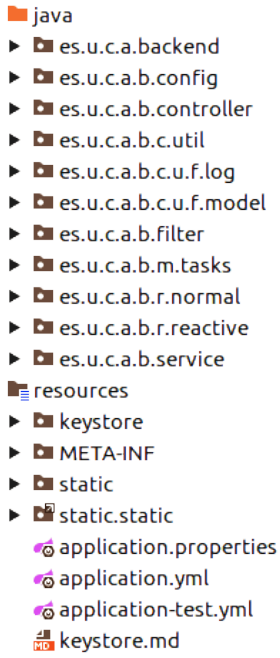


Figura 6.4: Estructura de paquetes: servidor

En la carpeta de recursos del servidor y del cluster se encuentra la configuración que será editable por el usuario, las claves de encriptación para proporcionar HTTPS y HTTP/2, y la interfaz optimizada para producción en el caso del servidor (carpetas *static*).

Se comenzará con un diagrama de paquetes para ver la estructura del código. Este se puede ver en la Figura 6.6: Diagrama de paquetes. La organización de estos se puede ver en las figuras 6.3, 6.4 y 6.5. La herramienta utilizada para generar los diagramas es PlantUML [21] con representación gráfica mediante yFiles [23].

En el diagrama de paquetes, se incluyen las dependencias de los 3 subproyectos. No se muestran todas las dependencias existentes porque Spring realiza inyección de dependencias para acelerar el desarrollo y esa característica se aprovecha mucho en el proyecto, incluso implementando una interfaz que realiza la ejecución del algoritmo y que según esten presentes unas clases u otras, Spring inyectará la ejecución local en el servidor o la ejecución remota en el clúster externo.

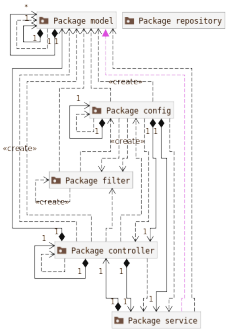


Figura 6.6: Diagrama de paquetes.

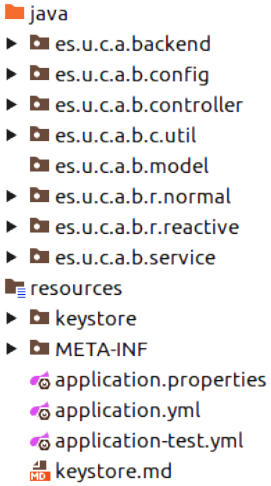


Figura 6.5: Estructura de paquetes: clúster

6.2. Diseño e implementación

6.2.1. Controller

El diagrama de clases del paquete de controladores, que es el principal, se encuentra en la Figura 6.7 y la Figura 6.8. Las dependencias no suele ocurrir entre

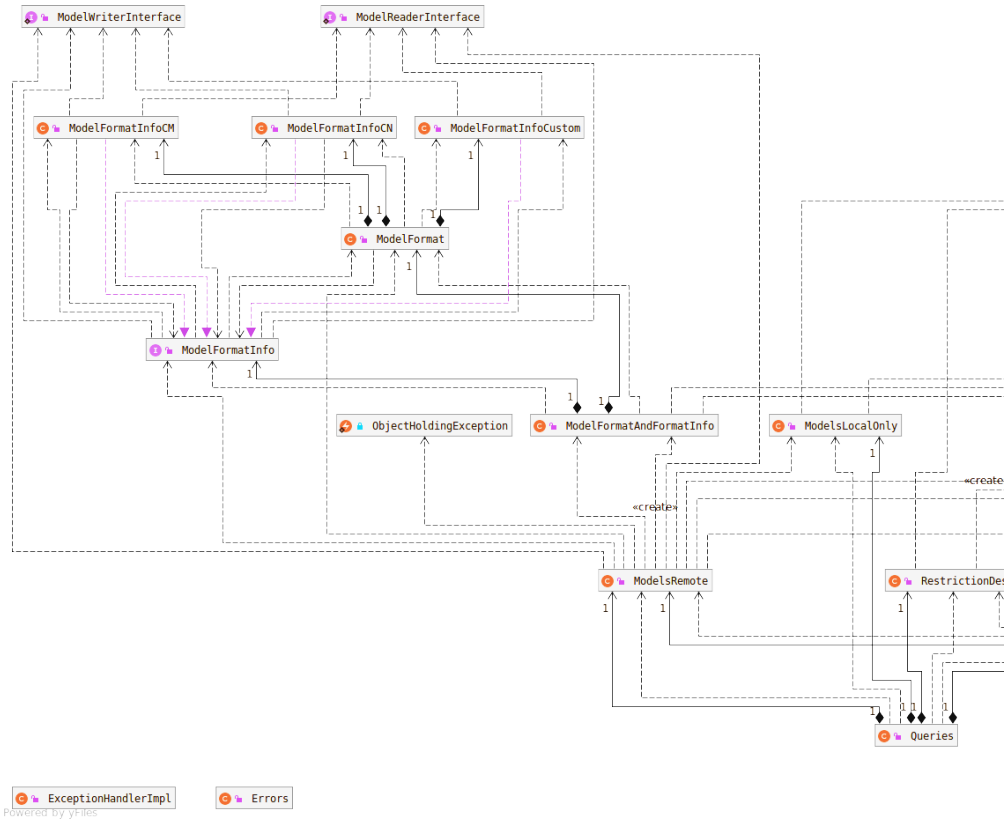


Figura 6.7: Diagrama de clases del paquete *controller*, parte 1.

clases del paquete controlador (ya que proporcionan servicios diferentes), sino que existen dependencias con el subpaquete de utilidades de los controladores, que contiene elementos como formatos de registros y modelos.

El diagrama contiene las jerarquías de registros y modelos (consultas) para facilitar el manejo de distintos formatos. También se muestran todos los controladores REST, cuyos métodos son los servicios disponibles documentados con OpenAPI [25] y protegidos por seguridad basada en roles, y usan transacciones de forma que se maneje correctamente la base de datos NoSQL (lo cual llevó a reemplazar llamadas a *exist()* y *count()* porque no están soportadas con tanta seguridad). La clase *Util* proporciona unos métodos usados para el filtrado avanzado de los modelos, permitiendo que cualquier método de búsqueda de los servicios implemente fácilmente este filtrado.

El diagrama de secuencia disponible en la Figura 6.9 y la Figura 6.10 muestra el método completo que actúa de servicio que ejecuta el algoritmo. Se puede comprobar como comienza con una comprobación y recuperación de log, modelo y restricciones que emite un error si no se encuentran.

Después se crea una tarea (el método λ), porque se trata de un proceso que puede durar mucho y no debe ser obligatorio mantener la conexión al servicio

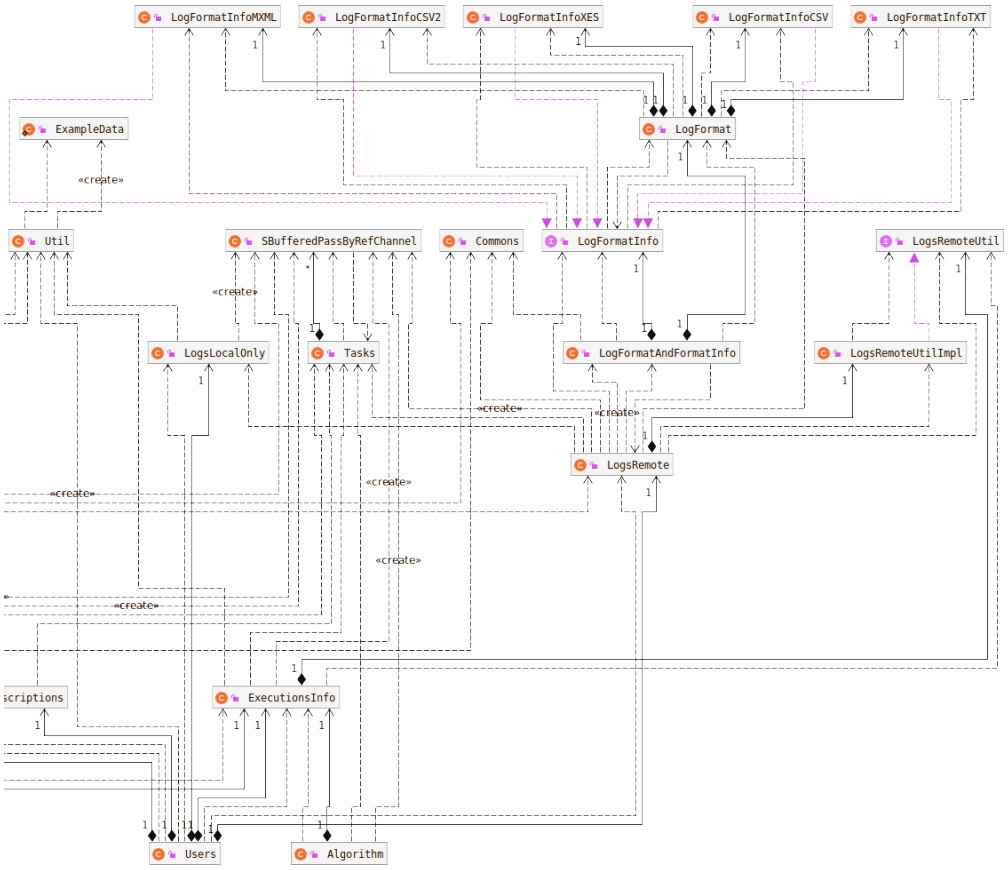


Figura 6.8: Diagrama de clases del paquete *controller*, parte 2.

abierta, sino que de esta forma se consiguen desacoplar completamente el servicio y el cliente, pero sin perder la capacidad de recibir un registro en tiempo real de las actualizaciones que ocurren en la tarea. El usuario se puede conectar y desconectar cuando quiera a otro servicio que le proporciona estas actualizaciones de tarea. Para completar la creación de la tarea se llama al método de registrar tarea, lo que finaliza la ejecución del servicio devolviendo un estado HTTP *202 Accepted*, indicando que se procesará en el fondo la tarea.

Mientras tanto, dentro de la tarea y de forma asíncrona, se realiza una inicialización de la misma, seguida de un *checkpoint* o punto de control y la ejecución (*apply*) del algoritmo, que dentro en cada actualización de progreso (λ siguiente al reply) informa al usuario y realiza otro punto de control.

Estos puntos de control son los que permiten la cancelación de cualquier tarea. Como las tareas se ejecutan en sus propios hilos y no sería correcto interrumpir el hilo cuando el usuario pulsa cancelar (no permitiría a las tareas terminar correctamente limpiando los objetos intermedios), se marca la tarea como cancelada y cuando el hilo llega al próximo puntos de control se parará la ejecución de la tarea, informando al usuario.

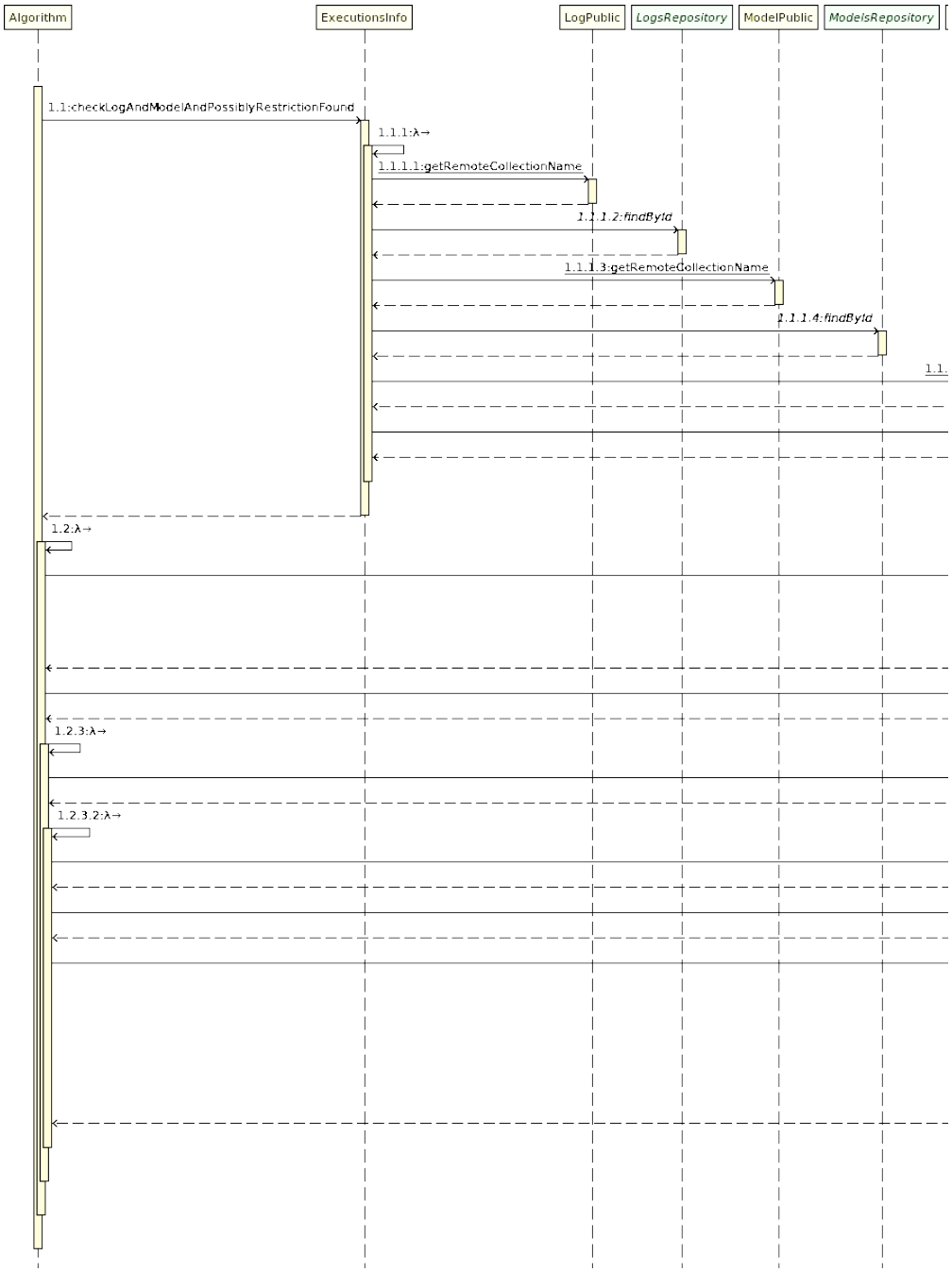


Figura 6.9: Diagrama de secuencia del servicio de algoritmo, parte 1.

El método *apply* es una abstracción del algoritmo implementado en el incremento anterior. Es decir, si el servidor se usa por defecto o se configura para su uso en local, invoca directamente al algoritmo; si en cambio se configuró para la ejecución en un clúster remoto, hace la llamada HTTP al servicio remoto implementado en el proyecto del clúster y también redirige los registros en tiempo real.

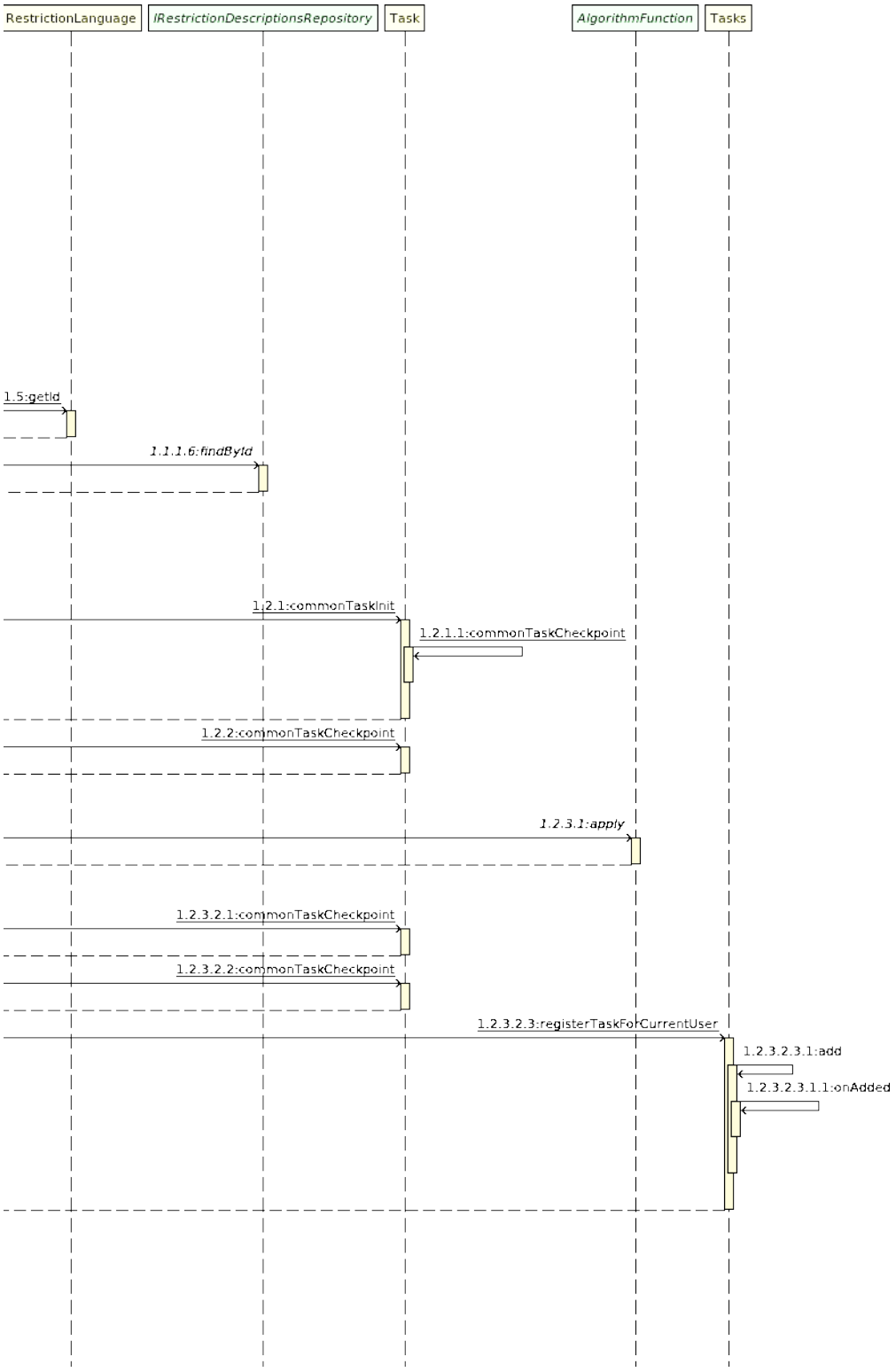


Figura 6.10: Diagrama de secuencia del servicio de algoritmo, parte 2.

6.2.2. Config

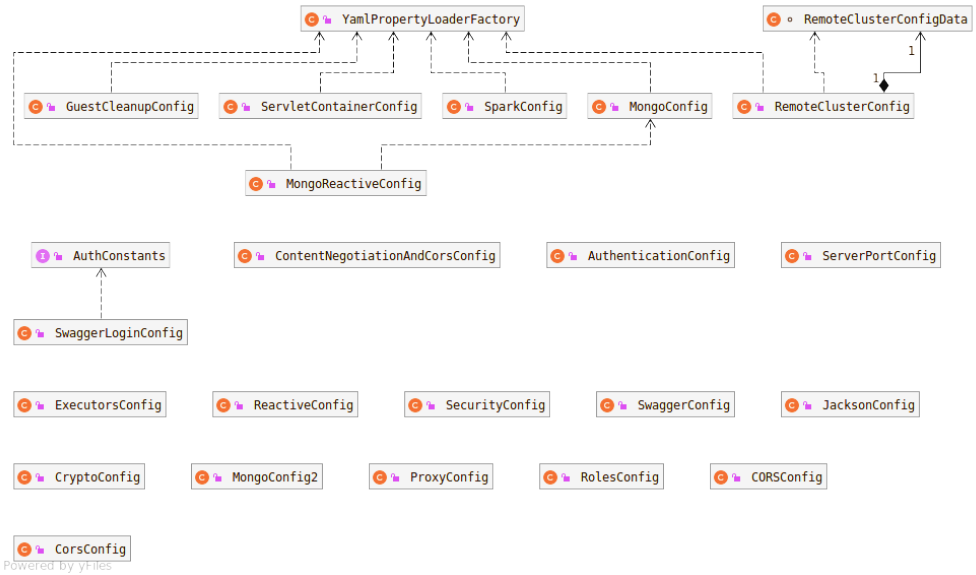


Figura 6.11: Diagrama de clases del paquete *config*.

El diagrama de clases del paquete de *config*, que realiza toda la configuración de Spring, se encuentra en la Figura 6.11. Solo se incluyen las dependencias básicas entre clases (implementación y extensión) para reducir el tamaño del diagrama. Existen pocas relaciones entre las clases porque la configuración se carga mediante sistemas de inyección de dependencias usado por Spring que no se puede representar.

Destaca la relación de algunas clases con *YamlPropertyLoaderFactory*, que permite la carga de configuración desde fichero *YAML*, que es más cómodo de mantener que los ficheros *properties*. La configuración incluye, entre otros, el tiempo de usuarios invitados, Spark, MongoDB (se configuran las bases de datos local y remota) y el clúster remoto para ejecución del algoritmo.

6.2.3. Filter

El diagrama de clases del paquete de *filter*, que implementa la autenticación (identificar a los usuarios) y autorización (delimitar los permisos mediante roles), se encuentra en la Figura 6.12. Estas son las funcionalidades de las clases *AuthenticationFilter* y *AuthorizationFilter* respectivamente. Existen los roles de usuario, moderador y administrador, cada uno con más permisos que el anterior (los moderadores pueden bloquear cuentas de otros usuarios y los administradores pueden realizar muchas más tareas sobre cuentas de otros usuarios, como

el borrado de toda la información, registros, consultas... asociados o el cambio de roles).

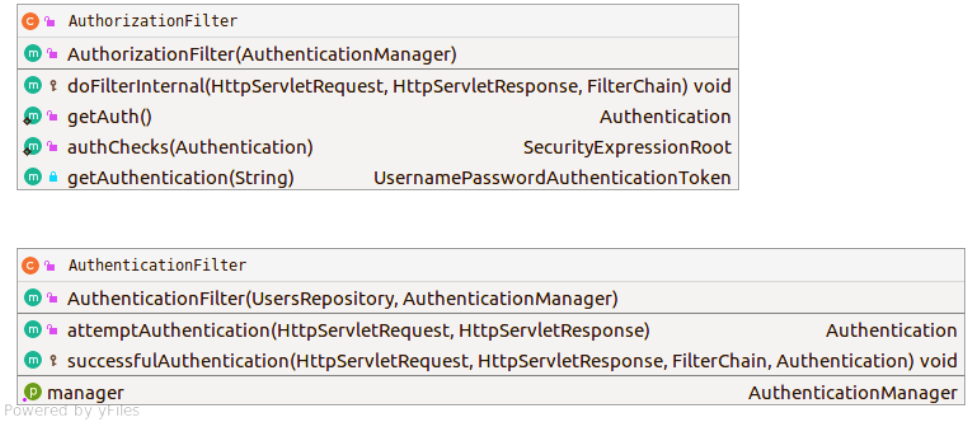


Figura 6.12: Diagrama de clases del paquete *filter*.

6.2.4. Model

El diagrama de clases del paquete de *model*, que representa toda la información asociada a los elementos con los que se trabaja, se encuentra en la Figura 6.13 y la Figura 6.14.

A la izquierda del diagrama se pueden ver las clases de manejo de tareas y de errores. Estas se abstraen usando el patrón Facade mediante la clase Tasks que es a la que se accede desde el paquete *controller*. Los errores se manejan de la misma forma en todos los servicios, lo que permite indicarlos en formato JSON con campos `HttpStatus status`, `String message`, `List<String> errors`, `HttpHeaders headers` y `Throwable cause`; que proporcionan mucha más información a los clientes para corregir los problemas.

El bloque central superior se encarga del manejo de registros y referencias a registros. Es importante mencionar que sólo se mantienen los metadatos necesarios para mejorar las interfaces sobre registros y modelos, mientras que el almacenamiento actual de ambos se realiza en la base de datos remota para optimizar la ejecución del algoritmo.

Debajo de este está la información de ejecución, que extiende de los parámetros de ejecución y tiene un atributo que es el resultado obtenido, si se ha ejecutado. También existe la clase *Model* para almacenar los metadatos de las consultas.

En el árbol de arriba a la derecha se representan todas las posibles restricciones, a partir de la interfaz IRestrictionDescription, lo cual permite que se extiendan las restricciones para permitir más lenguajes o incluso integrar

nuevos sistemas de restricciones fácilmente. Por último, el árbol de abajo a la derecha representa los usuarios de la aplicación.

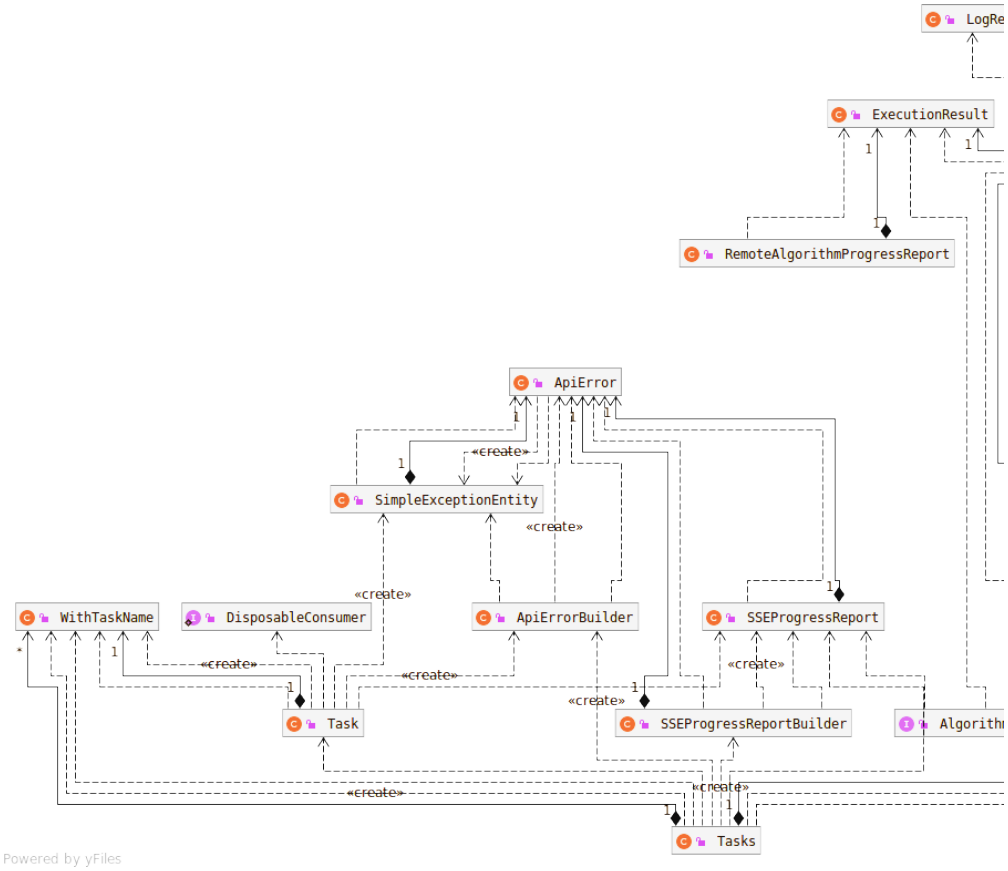


Figura 6.13: Diagrama de clases del paquete *model*, parte 1.

6.2.5. Repository

El diagrama de clases del paquete de *repository*, que define la interacción necesaria con la base de datos, la cual se implementa con clases autogeneradas en base a esta por *Spring*, se encuentra en la Figura 6.15. Las clases no están relacionadas entre sí y dan acceso a todos los requisitos de información planteados.

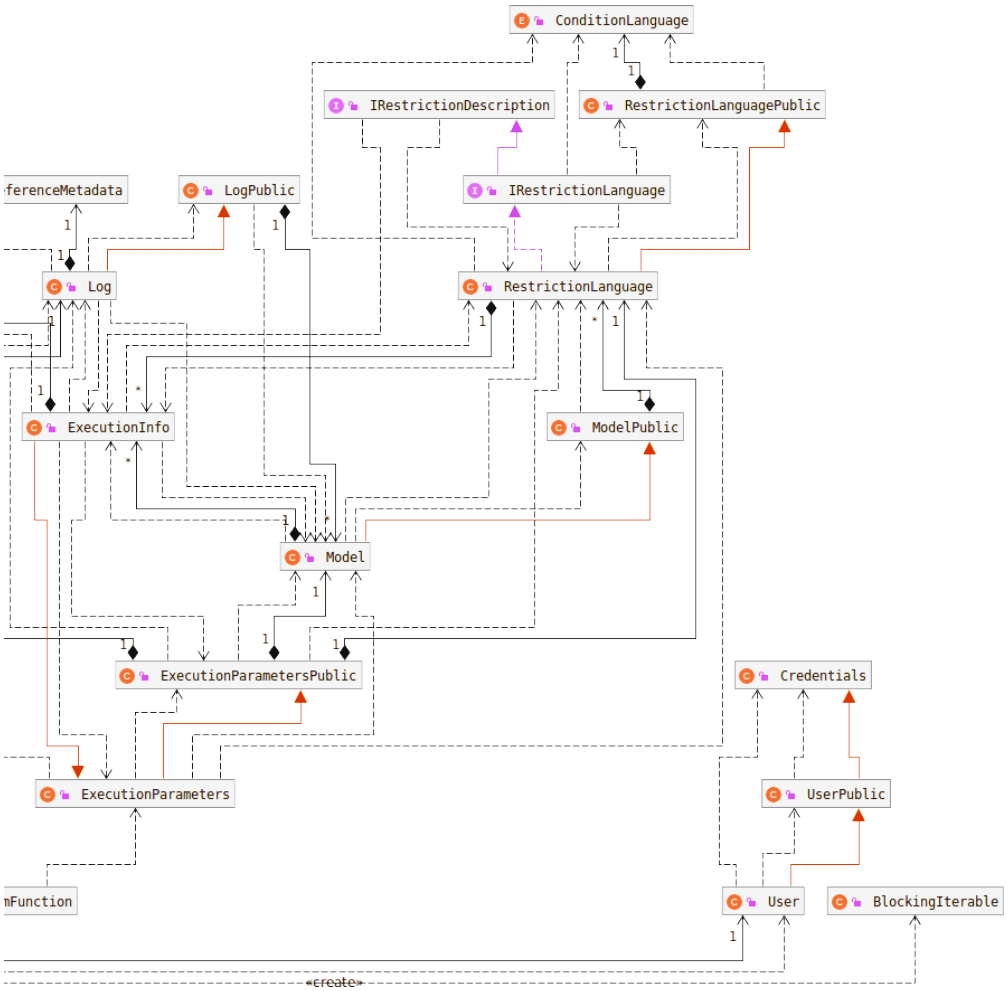


Figura 6.14: Diagrama de clases del paquete *model*, parte 2.

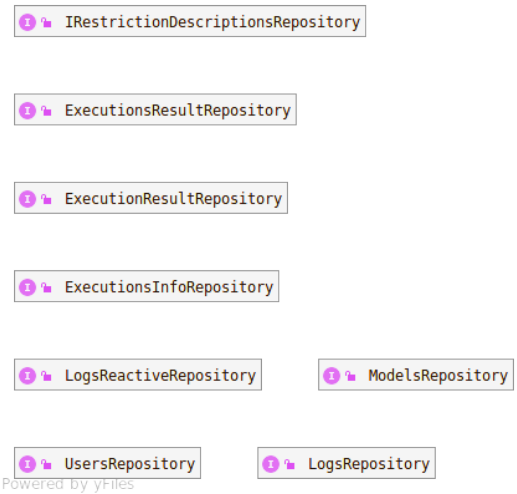


Figura 6.15: Diagrama de clases del paquete *repository*.

6.2.6. Service

El diagrama de clases del paquete de *service*, que implementa el servicio del algoritmo y el envío de correos y la obtención de detalles de usuarios, se encuentra en la Figura 6.16. Estas son las funcionalidades de las clases *AlgorithmService*, *EmailServiceImpl* y *UserAuthDetailsService* respectivamente.

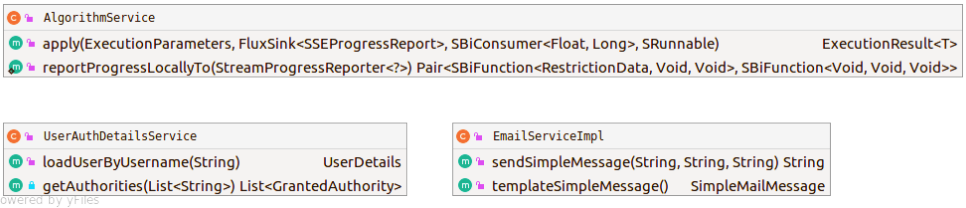


Figura 6.16: Diagrama de clases del paquete *service*.

6.2.7. Base de datos

En la Figura 6.17, la Figura 6.18, la Figura 6.19 y la Figura 6.20 se muestra el diseño de la base de datos. La representación del contenido de la base de datos se realizó transformando los datos contenidos al lenguaje *GraphQL* y se representa usando la página *GraphQL Voyager*[2], obteniendo un resultado similar a un diagrama de clases.

En la segunda columna se pueden ver las colecciones almacenadas en las dos bases de datos desarrolladas para este incremento. Las dos bases de datos se mantienen separadas porque una esta optimizada para su uso en el servidor (que guarda información directamente relacionada con la interfaz), y otra en el clúster (que puede mantenerse en la misma instancia de MongoDB, y resulta útil para que el clúster disponga de los datos localmente si se distribuye la base de datos). A la primera pertenecen las colecciones de usuarios, restricciones, información sobre registros, información sobre consultas, ejecuciones (configuración) y resultados; mientras que a la segunda pertenecen los contenidos de los registros y modelos en una colección para cada uno (los registros necesitan dos como ya se explicó en el incremento anterior).

En la tercera columna se pueden ver muchas de las estructuras necesarias para trabajar. Cuando se indica una estructura del tipo (ref, id), se trata de una referencia a un documento de la colección ref con el id indicado, optimizando el almacenamiento NoSQL.

[2]https://apis.guru/graphql-voyager/

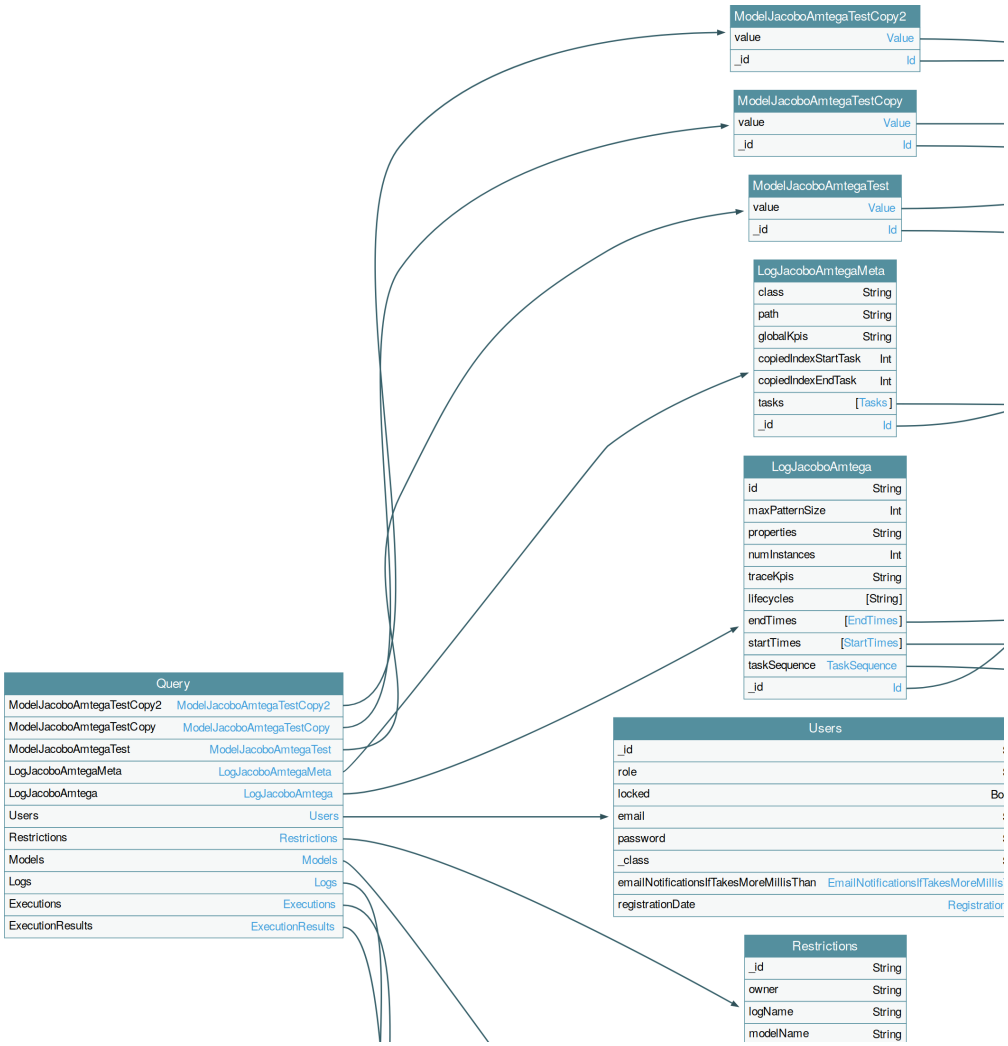


Figura 6.17: Diseño de la base de datos este incremento, parte 1.

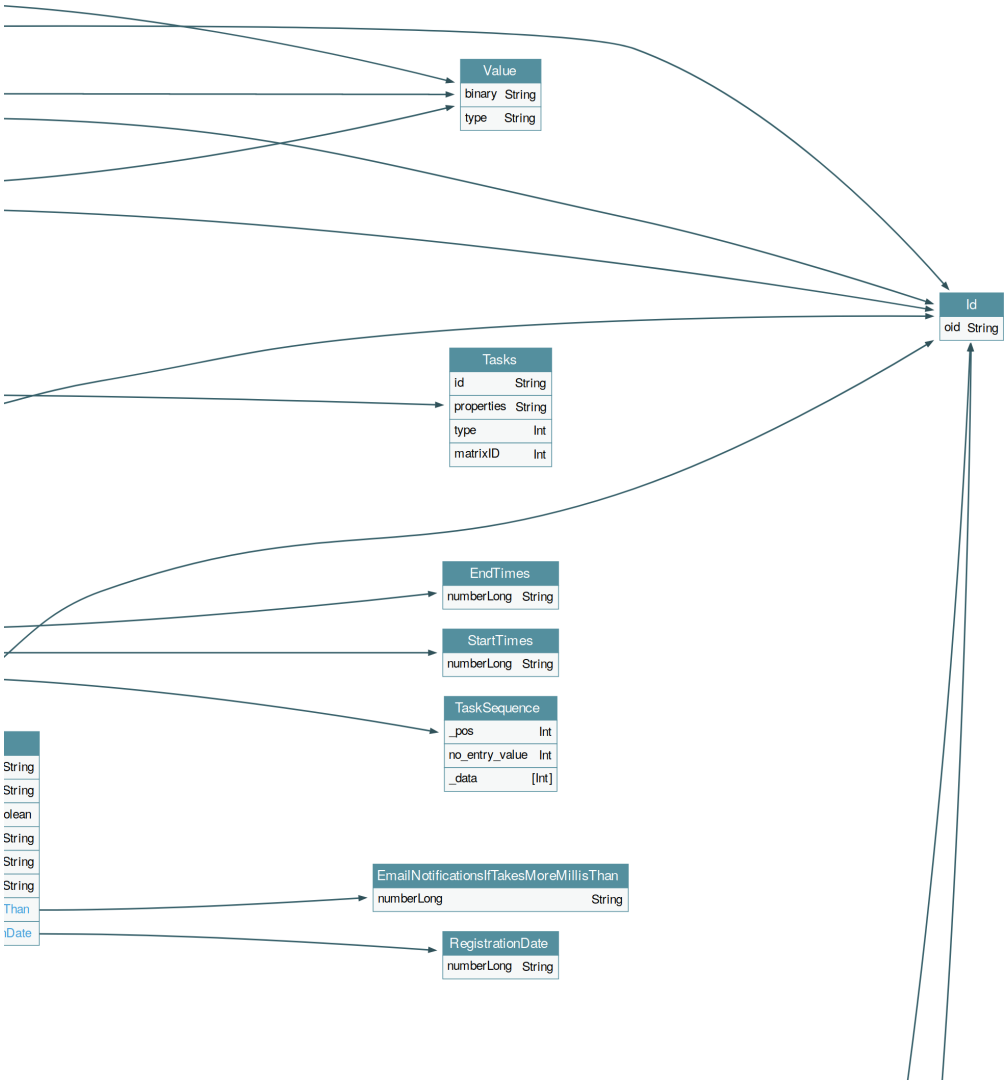


Figura 6.18: Diseño de la base de datos este incremento, parte 2.

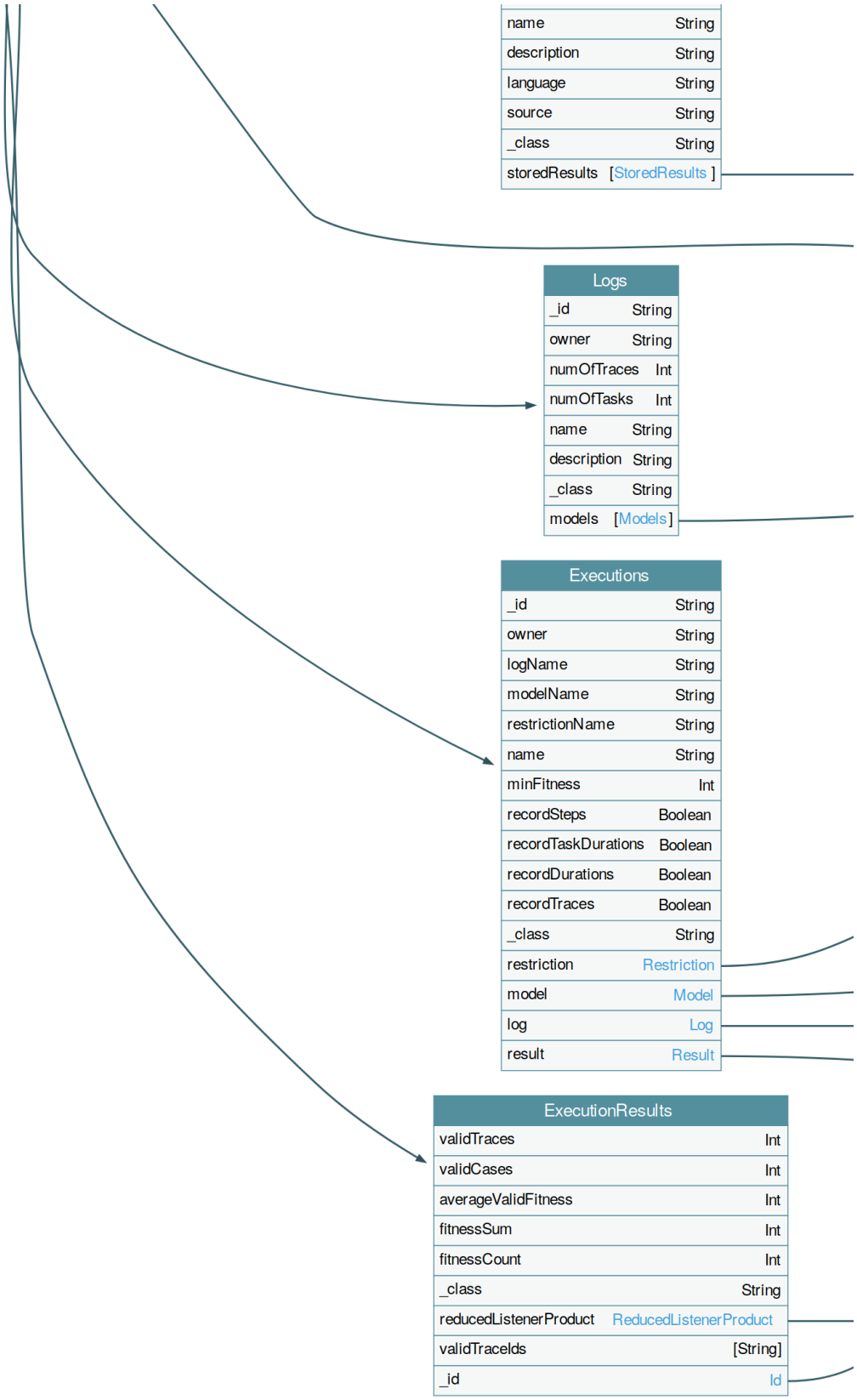


Figura 6.19: Diseño de la base de datos este incremento, parte 3.

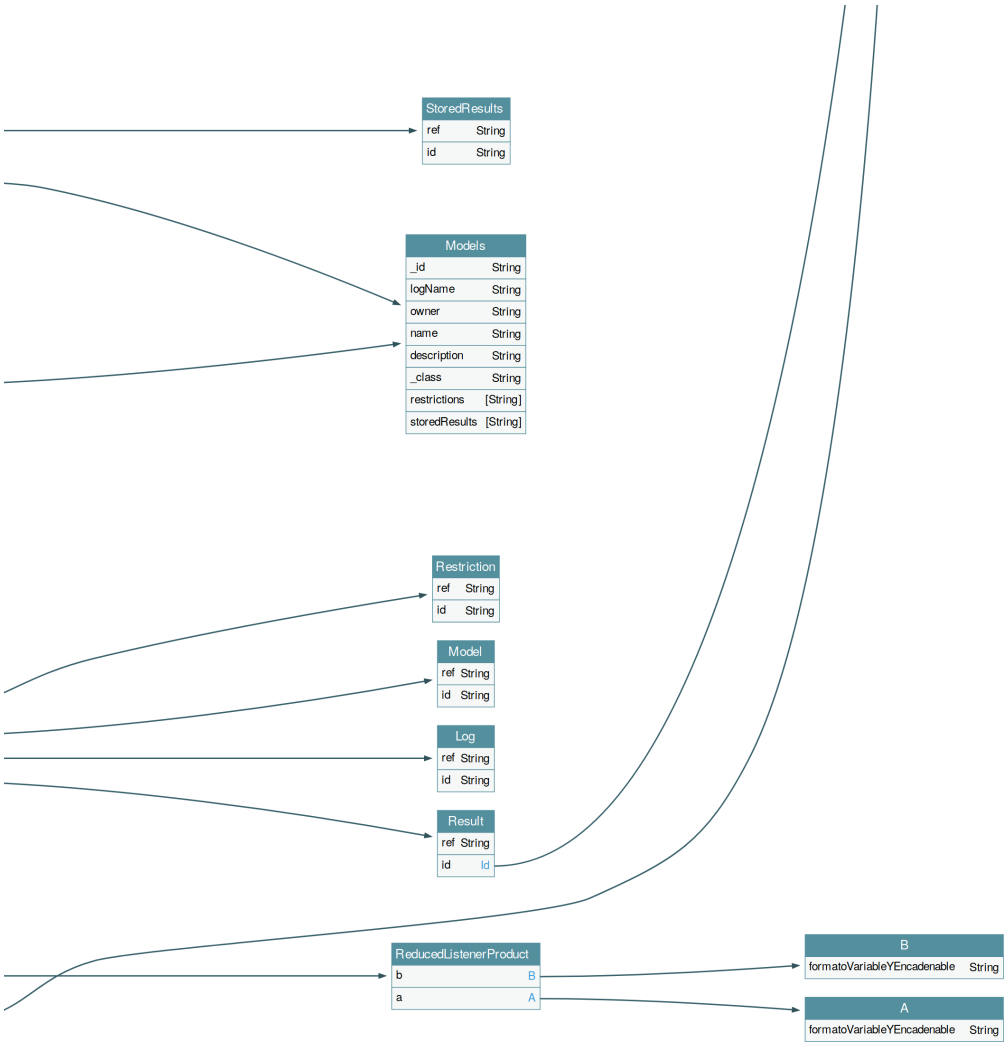


Figura 6.20: Diseño de la base de datos este incremento, parte 4.

6.3. Pruebas

6.3.1. Objetivos

Este incremento no necesita una etapa de prueba tan fuerte como el primero ya que se probará a medida que se desarrolla la interfaz que directamente llama a los servicios ofrecidos mediante un cliente autogenerado en base a la especificación OpenAPI [25]. Es necesario que todas las pruebas pasen y que exista una cobertura superior al 25 % en instrucciones para considerar el trabajo de este incremento realizado.

El objetivo principal es permitir que se pueda ejecutar el algoritmo de forma remota (lo cual implica la carga de usuarios, registros, consultas y configuraciones y la ejecución remota del mismo).

6.3.2. Diseño de pruebas y resultados

En la Tabla 6.1 se detallan las pruebas que dieron lugar a todos los casos de prueba, que verifican los algoritmos. La técnicas siempre es el uso de JUnit 5 para ejecutar los casos de prueba en cualquier orden, comenzando por la creación de un usuario de prueba que será borrado al final del caso de prueba, eliminando toda la información relacionada creada durante el caso de prueba. Solo existe un caso de prueba por cada prueba porque el código de este incremento ofrece servicios que usan el incremento anterior ya probado.

Nombre (ID)	testContextLoads() (PR2-1)
Objetivo	Comprobar que el servidor inicia correctamente.
Estrategia	Se comprobará que se inicia el sistema de *Spark*, se conecta a *MongoDB*, comprueba la disponibilidad del servicio de *email* y toda la configuración realizada.
Funcionalidad	Es la base de todas las funcionalidades, pero principalmente RFQ-9: Configurar y ejecutar el algoritmo.
Nombre (ID)	registrationWorksThroughAllLayers() (PR2-2)
Objetivo	Comprobar que se registran correctamente los usuarios.
Estrategia	Se comprobará que tras el registro se puede iniciar sesión con la cuenta y la base de datos contiene todos los datos del registro.
Funcionalidad	RFQ-14: Crear usuario o invitado.
Nombre (ID)	loginAndListLogs() (PR2-3)
Objetivo	Comprobar que se pueden listar los registros.
Estrategia	Se comprobará que tras iniciar sesión se pueden listar los registros, que en este caso devolverá la lista vacía con estado 200 OK.
Funcionalidad	RFQ-1: Listar registros, consultas, restricciones, configuraciones y resultados, usuarios y las actividades de los registros.
Nombre (ID)	uploadLog() (PR2-4)
Objetivo	Comprobar que el servidor importa registros correctamente.
Estrategia	Se comprobará que se tras iniciar sesión se puede subir un registro, y que este aparece correctamente en la base de datos.
Funcionalidad	RFQ-1: Listar registros, consultas, restricciones, configuraciones y resultados, usuarios y las actividades de los registros y RFQ-2: Importación y exportación de registros, consultas y restricciones, y exportación de resultados.
Nombre (ID)	uploadLogAndQuery() (PR2-5)
Objetivo	Comprobar que el servidor importa consultas correctamente.

Estrategia	Se comprobará que se tras iniciar sesión y subir un registro se puede subir una consulta asociada al registro, y que esta aparece correctamente en la base de datos.
Funcionalidad	RFQ-1: Listar registros, consultas, restricciones, configuraciones y resultados, usuarios y las actividades de los registros y RFQ-2: Importación y exportación de registros, consultas y restricciones, y exportación de resultados.
Nombre (ID)	uploadLogAndQueryAndExecute() (PR2-6)
Objetivo	Comprobar que el servidor ejecuta el algoritmo correctamente.
Estrategia	Se comprobará que se tras iniciar sesión y subir un registro y una consulta asociada se puede ejecutar el algoritmo sobre estos datos.
Funcionalidad	RFQ-1: Listar registros, consultas, restricciones, configuraciones y resultados, usuarios y las actividades de los registros, RFQ-2: Importación y exportación de registros, consultas y restricciones, y exportación de resultados y RFQ-9: Configurar y ejecutar el algoritmo.

Tabla 6.1: Pruebas diseñadas.

Los resultados se pueden ver en la Figura 6.21. Los tiempos obtenidos no son adecuados para medir el rendimiento porque en lugar de conectarse a las tareas para identificar el tiempo exacto en el que terminan se deja un tiempo fijo en el que se deben completar las tareas antes de realizar las comprobaciones adecuadas sin complicar el código de pruebas, y en la interfaz se experimentarán tiempos mucho más rápidos.

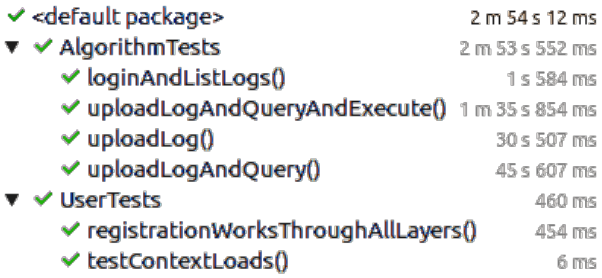


Figura 6.21: Resultados de casos de prueba.

6.3.3. Cobertura

Se pueden ver en la Figura 6.22 y la Figura 6.23, que indica que se comprueban el **31 % (4,274 de 13,571)** de las instrucciones de todo el código, entre muchos otros datos; superando el límite impuesto en los objetivos. Para más información se puede descargar el artefacto generado al ejecutar todas las pruebas, el cual incluye la cobertura de los contenidos de los paquetes y también

unas vistas de la cobertura de cada clase, en las cuales se muestran las líneas ejecutadas, no ejecutadas y en las que no se ejecutó algún tipo de salto.

Element	Missed Instructions	Cov.	Mi
es.usc.citius.alignments.backend.controller		25%	
es.usc.citius.alignments.backend.controller.util		20%	
es.usc.citius.alignments.backend.controller.util.formats.log		32%	
es.usc.citius.alignments.backend.config		57%	
es.usc.citius.alignments.backend.controller.util.formats.model		35%	
es.usc.citius.alignments.backend.model.tasks		41%	
es.usc.citius.alignments.backend.filter		86%	
es.usc.citius.alignments.backend.service		67%	
es.usc.citius.alignments.backend		37%	
Total	9,297 of 13,571	31%	60

Figura 6.22: Cobertura de los paquetes, parte 1.

ssed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
	18%	351	406	1,079	1,486	119	168	4	18
	26%	76	95	215	290	43	59	2	6
	7%	89	107	134	189	55	72	4	8
	19%	37	84	127	310	25	71	4	19
	15%	58	74	147	215	28	44	2	6
	22%	18	28	59	111	9	19	0	1
	58%	8	19	8	55	3	13	1	3
	50%	4	9	7	24	3	8	0	2
	n/a	1	2	2	3	1	2	0	1
0 of 736	18%	642	824	1,778	2,683	286	456	17	64

Figura 6.23: Cobertura de los paquetes, parte 2.

Capítulo 7

Incremento 3: interfaz gráfica

En este incremento se desarrolla la interfaz de la aplicación, la cual accede a los servicios ya desarrollados permitiendo un acceso a la herramienta cómodo para usuarios.

7.1. Arquitectura

7.1.1. Descripción

El front-end se desarrollará usando las tecnologías *React* [11], *Redux* [9], *JavaScript*, *Kotlin* [16] (transpilado a JavaScript) y *ANTLR4* [18], entre otras. El objetivo principal de la interfaz gráfica es permitir un acceso eficiente, reactivo y con una buena experiencia de usuario a todas las características del servidor web descritas anteriormente. Se escogió *yarn* sobre *npm* porque son compatibles mediante el uso de *package.json* para gestionar dependencias y *yarn* es más eficiente.

Además, se buscará que en la medida de lo posible sea accesible desde dispositivos móviles fácilmente (interfaz adaptativa). *React* permite el desarrollo de una interfaz muy eficiente y reusable para otros proyectos, y se optimizan sus componentes con *PureComponent*. Para mejorar la experiencia de usuario, se realizarán todas las peticiones web de forma asíncrona sin bloquear la interfaz y se mantiene una indicación de la vista actual en la URL de forma clara.

Kotlin, por las herramientas de las que dispone y por cercanía con Java (en el que se desarrolla el algoritmo y el servidor) y *JavaScript* permite escribir código de representación de grafos y analizar sintácticamente el lenguaje. Estas son tareas complejas para las que se puede reusar código de servidor en la interfaz y dar muy buenos resultados.

En el diseño hay elementos que no se detallan como el editor de texto, por lo que para hacerse una idea del mismo incluyo fotos del resultado en las figuras 7.1 y 7.2, que muestran el editor en árbol y una previsualización de grafo (que se carga desde código fuente desde la propia interfaz o si no está disponible el código fuente desde una representación *JSON* del modelo descargada del

servidor y procesada en la interfaz).

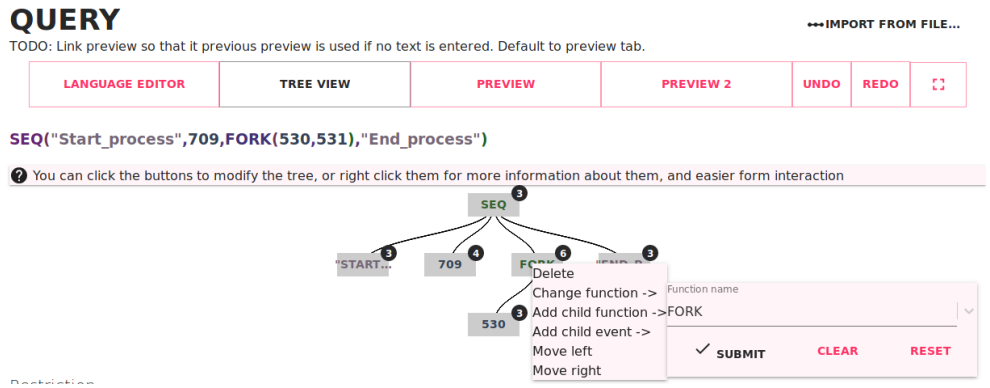


Figura 7.1: Editor: árbol sintáctico editable (el número son las acciones disponibles por nodo)

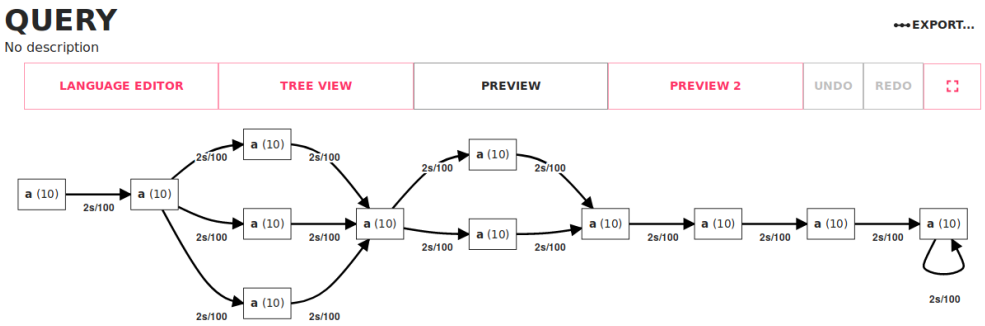


Figura 7.2: Editor: previsualización 1

Existe un bloque de código que consisten en el cliente autogenerado a partir de la descripción *OpenAPI* [25] proporcionada por el servidor. El cliente se autogenera en *TypeScript* que se compila a *JavaScript*. Cualquier desarrollador puede usar esta característica para acelerar el desarrollo de clientes en muchos lenguajes.

El bloque inferior a ese es el sistema de temas incorporado a la interfaz, para que cada usuario use el que le parezca más cómodo. Estos son globales (hasta los grafos se adaptan al tema), afectan a los colores y a los elementos (padding de botones, etc.), y permiten el cambio instantáneo sin perder la página. Este se guarda entre sesiones (como muchos otros datos y configuraciones, gracias a la integración de *Redux* realizada).

El resto de la interfaz habría que detallarla por vista para poder explicarla y se alargaría demasiado esta memoria. En su lugar explico los directorios en el siguiente apartado.

7.2. Diseño e implementación

El diseño incluye enlaces entre las distintas páginas (indicados con un símbolo), pero al incluirlo en este PDF se pierden, por lo que para interactuar con el proyecto se recomienda acceder al fichero de diseño V1 adjunto para poder usar estos enlaces y acercarse más a la experiencia del usuario de la aplicación.

En el diseño se incluyen anotaciones en amarillo que explican alguna interacción que no se podría describir con la herramienta de diseño utilizada (Balsamiq Mockups [5]). El diseño se encuentra en las figuras 7.4 y 7.5.

Los directorios registros, *models*, *restrictions* y *executioninfos* son modulares (cada uno tiene una parte específica por tratar con elementos distintos), los cuales manejan mediante tablas estos elementos, permitiendo buscarlos (filtrarlos, ordenarlos, cambiar de página y el tamaño de página), crear/importar nuevos, seleccionarlos accediendo a la siguiente tabla, exportarlos, borrarlos y actualizarlos. Todos están presentes en la interfaz vieja, mientras que en el rediseño que se explicará en apartados siguientes sólo se mantiene el uso de registros y *models*, mientras que los otros se simplificaron en el directorio *queries*, el cual permite ver y editar las consultas (modelo + restricción + parámetros de ejecución) y ver, guardar un registro a partir de las trazas aceptadas y exportar los resultados.

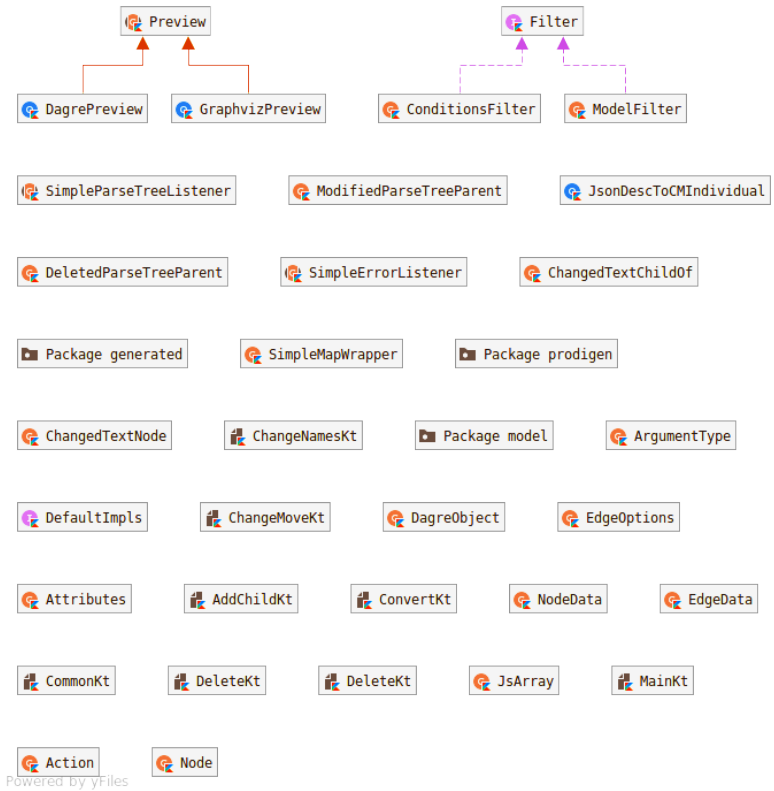


Figura 7.3: Clases del analizador sintáctico y editor de ambos lenguajes.

El directorio *editor* implementa el editor de código para ambos lenguajes desarrollados, formado por los módulos de editor textual, editor en árbol y previsualizaciones de la interpretación del lenguaje. Este paquete depende en el proyecto Kotlin [16] que usa ANTLR [18] para realizar el análisis sintáctico y ha sido compilado a JavaScript. El diagrama de clases para el proyecto del cual depende este código se muestra en la Figura 7.3. En este no se detectaron todas las dependencias entre clases por usar Kotlin. Se puede ver que existen dos previsualizaciones (mediante Graphviz y Dagre). El paquete *model* es una traducción del paquete *model* del incremento 2 a Kotlin, y ProDiGen la traducción de parte de la biblioteca base a Kotlin. El paquete *generated* es el analizador sintáctico generado. La gran mayoría de clases se encargan de facilitar el trabajo con árboles, implementando todas las 8 acciones disponibles para el lenguaje de modelos y mejorando la visualización de ambos árboles a partir del análisis sintáctico. En el editor en árbol se genera una cache de menús de contexto cuando se están generando los nodos, ya que estos tienen un peso importante en el rendimiento si se generan en cada clic.

El directorio *reducers* implementa las funciones que permiten el cambio de estado en Redux [9] interpretando acciones lanzadas por la interfaz. El directorio *themes* implementa los varios temas disponibles. El directorio *ts* contiene el código del cliente autogenerado en *TypeScript*, compilado a *JavaScript* en el directorio *tsout*. Los directorios *util*, *containers* y *components* contienen utilidades, componentes que contienen a otros y otros componentes; que son usando en distintos puntos de la aplicación.

Al tratarse de una *Single-Page Application (SPA)*, se puede hacer muy pesado descargar la interfaz entera cuando sólo se ven algunas vistas de la misma y en momentos distintos. Entonces se configura *webpack* para generar *chunks* de código que se aproximan a vistas de la interfaz (o componentes pesados como el editor de código). Estos chunks se sirven eficientemente gracias a HTTP/2 y se solicitan de forma asíncrona, mostrando un componente de carga en la posición que se está cargando. Todas las peticiones al servidor también son asíncronas, y se tratan de realizar en paralelos siempre que sea necesario realizar más de una petición.

La interfaz se optimiza en una compilación de producción que permite que se ejecute mucho más rápido y ocupe mucho menos para mejorar la experiencia de usuario. A parte de la configuración por defecto de producción de *webpack*, se añade el minimizador *TerserPlugin* configurado para máximo rendimiento en esta aplicación. Esta tarea está configurada para realizarse en el Apéndice D, por lo que se genera un artefacto en cada compilación que consiste en la interfaz optimizada.

7.3. Parada del incremento

Al realizar la prueba del cuestionario SUS con las preguntas de la Tabla 7.1

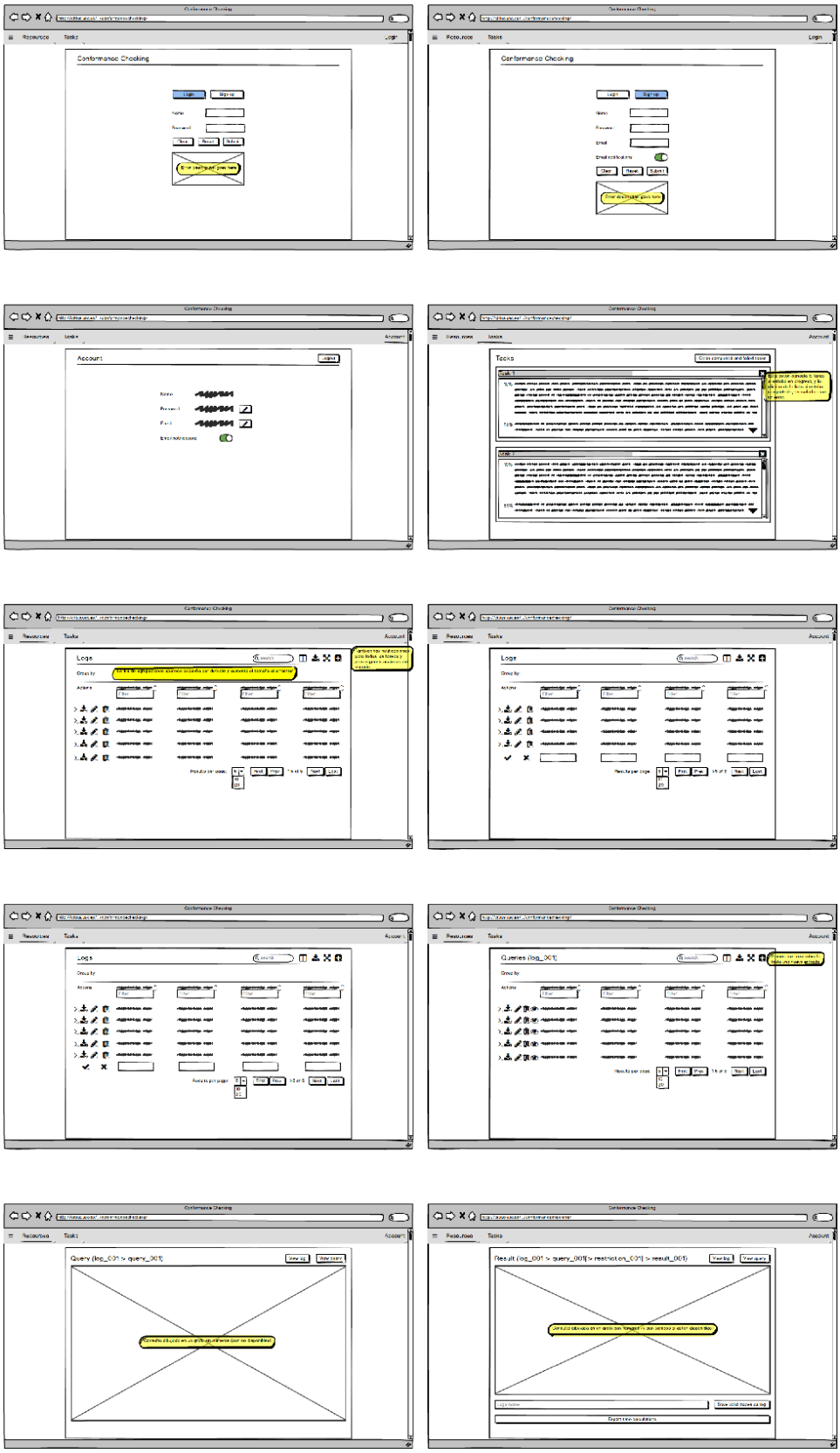


Figura 7.4: Diseño de la interfaz gráfica

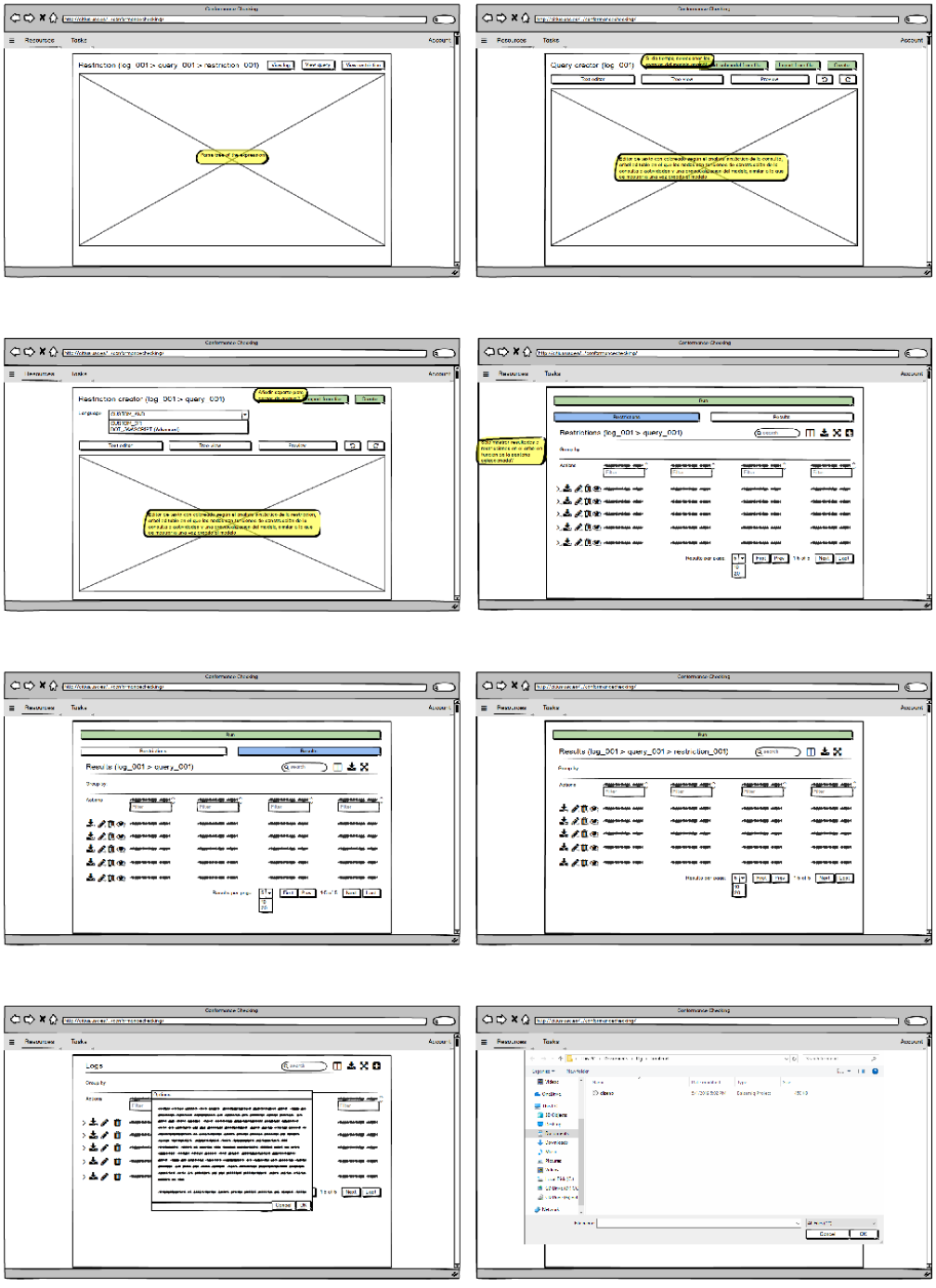


Figura 7.5: Diseño de la interfaz gráfica 2

se obtuvieron los resultados de la Tabla 7.2. Estos no son los adecuados para superar la prueba de validación propuesta en la sección de requisitos, por lo que es necesario un rediseño que se describe en el siguiente apartado.

#	**Pregunta**
1	La aplicación me ha resultado útil.
2	No encontré lo que buscaba.
3	Recomendaría su uso para trabajar en *conformance checking*.
4	Creo que la interfaz es difícil de usar.
5	La visualización de la consulta es cómoda.
6	No sé dónde introducir restricciones.
7	Conseguí filtrar, ordenar, cambiar el tamaño de las páginas y cambiar de página en las búsquedas.
8	Encontré el sitio innecesariamente complejo.
9	La interfaz ha cumplido con mis expectativas
10	No volvería a acceder a la aplicación de no ser la única opción.

Tabla 7.1: Cuestionario SUS.

Usuario	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Usuario 1	5	2	5	3	2	2	5	4	5	2
Usuario 2	5	2	4	3	3	2	5	3	5	1
Resultado	**7.5**									

Tabla 7.2: Resultados cuestionario SUS.

Capítulo 8

Incremento 3: interfaz gráfica v2

Una vez desarrollado el incremento de la interfaz (excepto la pestaña de cuenta y las visualizaciones específicas de los recursos, ya que no eran necesarias para el funcionamiento completo de la aplicación), se lo mostré a los tutores. Además, al realizar la validación con cuestionarios SUS a usuarios recibí una puntuación menor a la mínima indicada para aceptar la aplicación (de 7.5 sobre 10), por lo que fue necesario un rediseño. De las sugerencias que se recibieron y decidieron aplicar tanto de usuarios como de tutores destacan las siguientes.

1. Les pareció que la interfaz era demasiado compleja para los usuarios que podría tener. Para mejorarla, se decidió unir las ideas de consulta, restricción, configuración de algoritmo y resultado en un mismo elemento (al que llamamos consulta) y limitando el número de restricciones, configuraciones y resultados a 1 por cada consulta, de forma que se simplifique la interacción. Los cambios necesarios serían:

   a) El servidor sigue ofreciendo estos servicios por separado para permitir el desarrollo de aplicaciones más complejas, pero deberá ser optimizado para trabajar con estos conceptos unidos en menos peticiones REST.

   b) Las páginas de creación de estos elementos se combinan en una sola.

   c) Se puede visualizar cada consulta con su restricción (opcional), configuración de algoritmo y su resultado (opcional) en una sola página.

   d) Se debe permitir la edición de consultas ahora que tratan tantos conceptos, en lugar de permitir subir nuevos conceptos por separado. Esta también se realiza en la misma página que las restricciones y las ejecuciones.

2. Se quiere poder evitar el inicio de sesión para realizar consultas directamente sobre los servicios ofrecidos. Para imponer estas limitaciones que no tendrían los usuarios normales, se implementa:

   a) Prohibir al usuario invitado el almacenamiento de datos de forma permanente (tiempo configurable).

   b) Obligarlo a mantener la sesión abierta si quiere obtener resultados de una ejecución en progreso.

   c) Borrar el usuario invitado en un determinado tiempo (configurable).

3. Añadir un árbol de navegación a la izquierda para seleccionar registros y consultas rápidamente. Para evitar que este requiera cargar todos los elementos del usuario, los datos disponibles en las tablas páginadas. Esto ayudará a proporcionar el contexto de las ejecuciones del algoritmo y el acceso rápido entre consultas y registros.

4. Modificar la previsualización automática de consultas a partir del lenguaje para usar la biblioteca dagre-d3 en lugar de graphviz, ya que dagre-d3 se trata de una herramienta más moderna pensada para la web.

Al usar un ciclo de vida del proyecto por incrementos, implicó realizar un rediseño, implementación y prueba de una parte de la interfaz de usuario. Se decidió permitir al usuario usar ambas interfaces implementadas con un sencillo interruptor en la página, usando por defecto la nueva interfaz.

8.1. Diseño e implementación

El diseño incluye enlaces entre las distintas páginas (indicados con un símbolo), pero al incluirlo en este PDF se pierden, por lo que para interactuar con el proyecto recomiendo acceder al fichero de diseño V2 adjunto para poder usar estos enlaces y acercarse más a la experiencia del usuario de la aplicación.

En el diseño se incluyen anotaciones en amarillo que explican alguna interacción que no se podría describir con la herramienta de diseño utilizada (Balsamiq Mockups [5]). El diseño se encuentra en las figuras 8.1 y 8.2.

Las diferencias de implementación ya se explicaron en el capítulo anterior. La interfaz tras el rediseño necesita menos de 5MB para descargar toda la parte JavaScript de la interfaz (13,6 MB si se incluyen fuentes y otros ficheros de dependencias externas), mientras que sólo se van a descargar las vistas que es necesario renderizar, por lo que está muy optimizado. Además, el servidor ofrece compresión *gzip* y HTTP/2 (si se actualiza la dependencia de *Spark* para poder usar Java 9 o superior) si el cliente los soporta, lo que reduce aún más el tiempo de carga de la interfaz. Los *chunks* o bloques de código que se cargan por separado sólo cuando son necesarios son:

- 808K main.js
- 8,0K Account.js
- 8,0K Account AccountEdit.js
- 8,0K AccountEdit.js
- 4,0K AutoTask.js
- 8,0K CMIndividualDagrePreview.js
- 8,0K CMIndividualDagrePreview GraphvizPreview.js
- 16K CodeEditor.js
- 24K ExecutionInfoResourcesWrapper.js
- 2,2M GraphvizImport.js
- 8,0K GraphvizPreview.js
- 12K Languages.js
- 4,0K Languages RestrictionResourcesWrapper.js
- 16K LoginSignup.js
- 944K LogResourcesWrapper.js
- 908K ModelResourcesWrapper.js

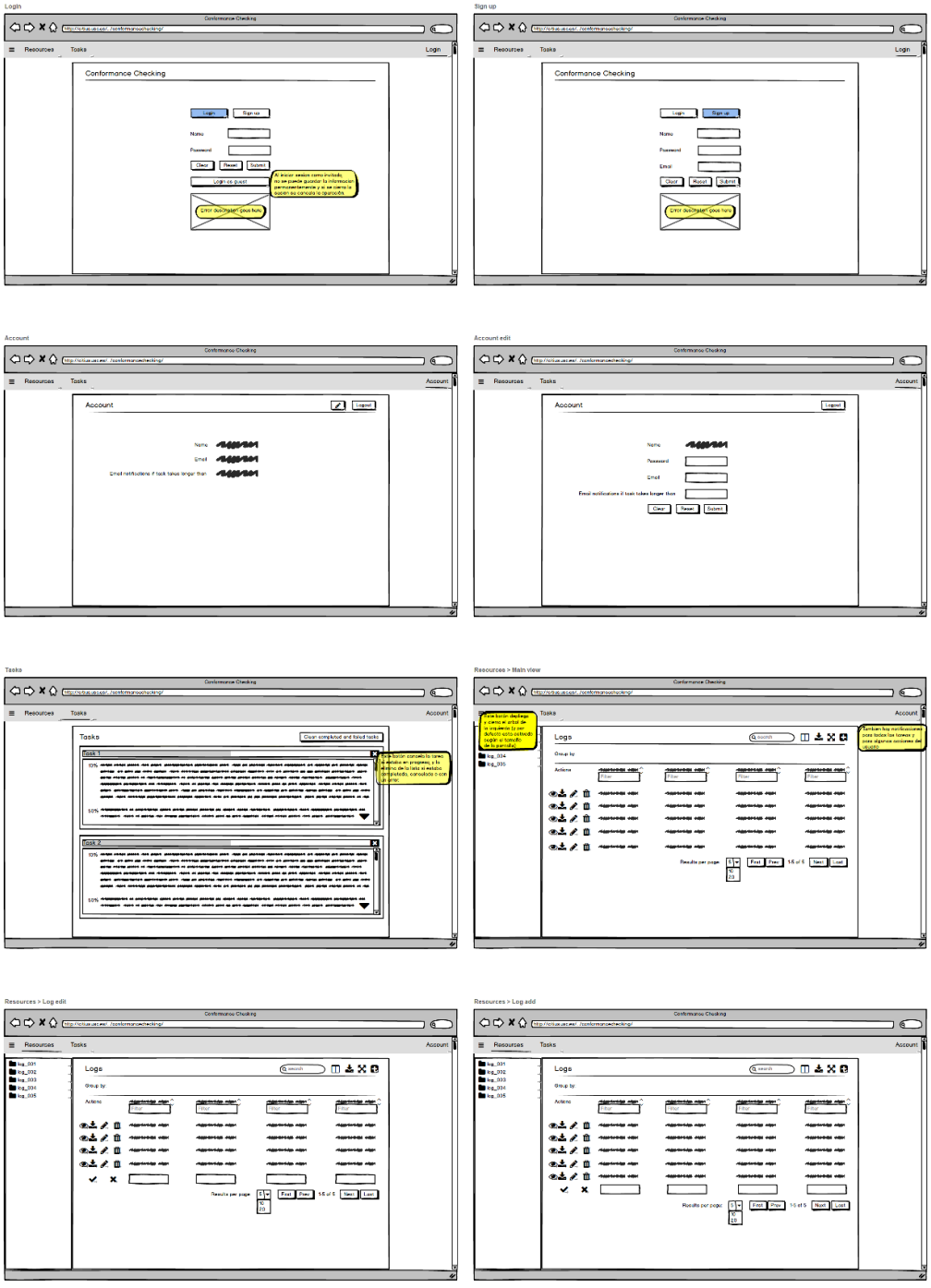


Figura 8.1: Rediseño de la interfaz gráfica

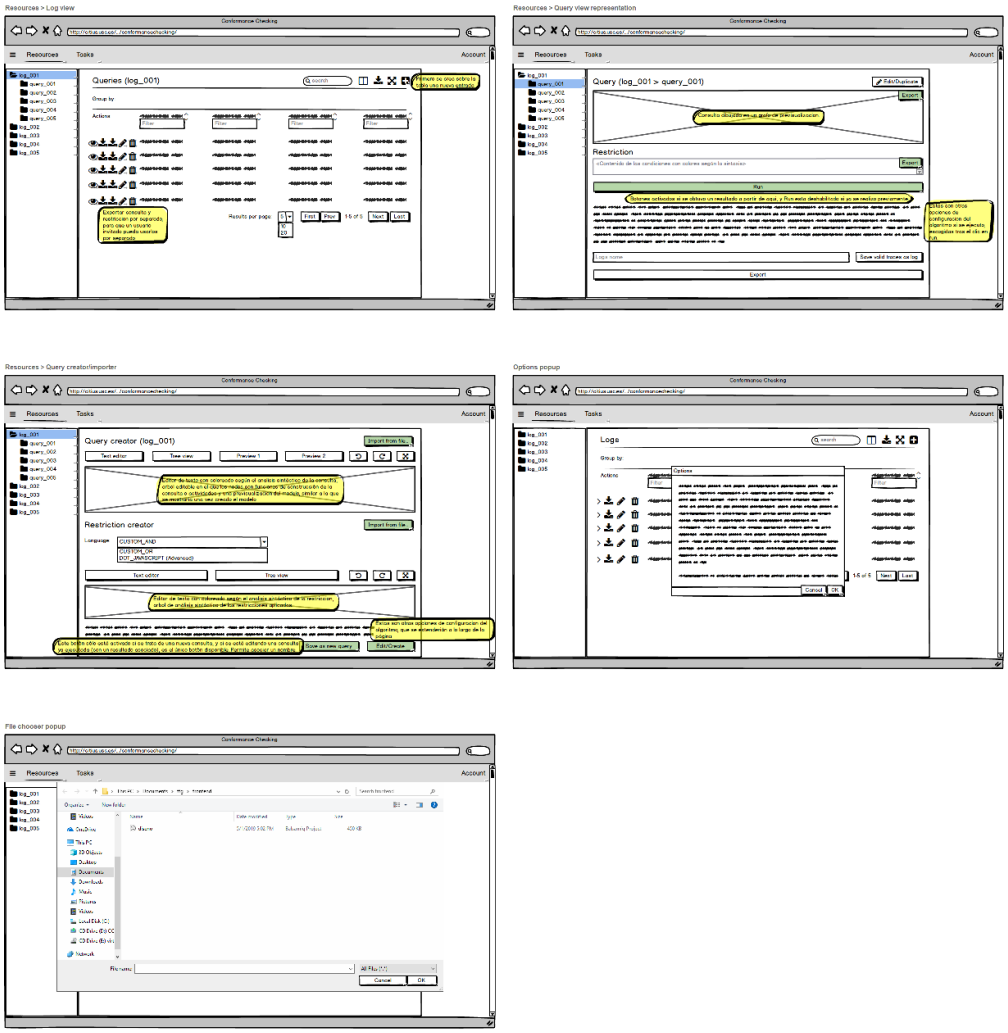


Figura 8.2: Rediseño de la interfaz gráfica 2

- 424K QueryEditor.js
- 72K QueryViewer.js
- 16K ResourcesSidebar.js
- 24K RestrictionResourcesWrapper.js
- 8,0K Task.js
- 4,0K Tasks.js
- 20K TextEditor.js
- 840K TreeEditor.js

Además, todas las dependencias se mantienen en bloques de código separados también, de forma que en caso de una actualización de alguna de ellas, la cache del navegador permitiría necesitar la descarga de la mínima parte del código, mejorando el tiempo de carga de la interfaz. La dependencia más pesada con diferencia es la de Graphviz, que se encarga de la previsualización del grafo secundaria y es cargada sólo si se accede a la pestaña asociada a este grafo del editor.

8.2. Pruebas

8.2.1. Objetivos

Este incremento no necesita una etapa de prueba tan fuerte como el primero ya que se probará a medida que se desarrolla la interfaz que directamente llama a los servicios ofrecidos. Es necesario que todas las pruebas pasen y que exista una cobertura superior al 25 % en instrucciones para considerar el trabajo de este incremento realizado. Además es necesario superar el cuestionario SUS con un resultado mayor a 8.

8.2.2. Cuestionario SUS

Al realizar el rediseño se volvió a realizar el cuestionario SUS a los mismos usuarios para comprobar que se mejoró la aplicación, y los resultados se muestran en la Tabla 8.1

Usuario	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Usuario 1	5	1	5	1	4	1	5	2	5	1
Usuario 2	5	1	5	2	5	1	5	2	5	1
Resultado	**9.5**									

Tabla 8.1: Resultados cuestionario SUS tras rediseño.

8.2.3. Diseño de pruebas y resultados

Se han generado 3 *suites* o conjuntos de pruebas que se ejecutan automáticamente en el CI/Entrega Continua (CD) al igual que todos los demás incrementos: las pruebas basadas en Redux, las unitarias que comprueban componentes React concretos y las de integración de toda la aplicación. Se incluyen pruebas de accesibilidad de la aplicación mediante Axe[1]. Las pruebas que usan Redux también son de integración porque comprueban cómo manejan todos los módulos de la interfaz los cambios de estado globales.

[1] https://www.deque.com/axe/

El uso de Redux también tiene la ventaja de que se pueden probar estados internos de la aplicación (iniciada sesión con esta cuenta, usando este tema, en esta URL (equivalente a en esta vista) y cualquier otra configuración que se esté manejando con Redux). Entonces aprovecho esta característica de la tecnología.

Para que las pruebas sean unitarias, se ha bloqueado el cliente si la variable global TEST está activada, de forma que no se intenten hacer consultas al servicio y se devuelvan datos *mockeados*.

En la Tabla 8.2 se detallan las pruebas que dieron lugar a todos los casos de prueba. La técnicas siempre es el uso de jest para ejecutar los casos de prueba en cualquier orden, generando los reportes con *jest-html-reporter*, *jest-html-reporters* y *jest-stare* (además de reportes de cobertura) y renderinzando componentes *React* con *enzyme*, los cuales están disponibles como artefactos en cada ejecución del CI en GitLab.

Unit tests suite	
Nombre (ID)	should start test framework (PR3-1)
Objetivo	Comprobar que el sistema de prueba se ha configurado correctamente.
Estrategia	Se comprobará que se ejecuta la prueba y se reciben todos los informes configurados.
Funcionalidad	Es la base de todas las funcionalidades de las pruebas siguientes.
Nombre (ID)	should shallow render the main component (PR3-2)
Objetivo	Comprobar que el componente principal de la aplicación renderiza correctamente (sin renderizar los componentes a los que hace referencia).
Estrategia	Se comprobará que se renderiza el componente principal sin definir propiedades y sin lanzar errores, con y sin realizar snapshots que comprueban que el contenido es tal cual el renderizado en la última ejecución.
Funcionalidad	Es la base de todas las funcionalidades de las pruebas siguientes.
Redux suite	
Nombre (ID)	app startup (PR3-3)
Objetivo	Comprobar que la interfaz inicia correctamente.
Estrategia	Se envían todos los eventos de Redux que se emitirían en un inicio correcto de la aplicación, renderizando la aplicación completa en cada estado intermedio y comprobando que no ocurren errores.
Funcionalidad	Es la base de todas las funcionalidades de las pruebas siguientes.
Nombre (ID)	should navigate through all pages (PR3-4)

Objetivo	Comprobar que se pueden acceder a todas las funcionalidades de la interfaz.
Estrategia	Se envían todos los eventos de Redux que se emitirían en una navegación entre páginas de la aplicación, renderizando la aplicación completa en cada estado intermedio y comprobando que no ocurren errores. Se escribirá un caso de uso por cada página a la que acceder para identificar más fácilmente los errores.
Funcionalidad	Todas, ya que la interfaz permite acceder a todas los requisitos funcionales de la aplicación.
Integration suite	
Nombre (ID)	should mount the whole app (PR3-5)
Objetivo	Comprobar que se monta y renderiza correctamente el componente de la aplicación incluyendo todos los que hace referencia.
Estrategia	Se comprobará que se renderiza el componente principal sin definir propiedades y sin lanzar errores, con y sin realizar snapshots que comprueban que el contenido es tal cual el renderizado en la última ejecución.
Funcionalidad	Es la base de todas las funcionalidades comprobadas por otras pruebas.
Nombre (ID)	should be accessible to all users (PR3-6)
Objetivo	Comprobar que la aplicación es accesible.
Estrategia	Se comprobará usando la herramienta jest-axe que asegura que al menos una aplicación es un 30 % accesible (realizando comprobaciones como que las imágenes y botones dispongan de texto explicativo ya que no puede realizar comprobaciones más complejas). Para completar esta prueba se realizó el estudio mediante un cuestionario SUS disponible en la Subsección 8.2.2.
Funcionalidad	Es la base de todas las funcionalidades comprobadas por otras pruebas.

Tabla 8.2: Pruebas diseñadas.

En la Figura 8.3 se puede ver el resumen de pruebas ejecutadas, indicando el resultado y el tiempo necesario. Los primer nivel del árbol indica la suite a la que pertenecen las pruebas, el segundo son las pruebas (y los casos de prueba si coincide que solo hay un caso), y en el tercer nivel están los casos de prueba.

8.2.4. Cobertura

La cobertura de todas los casos de pruebas se muestra en la Figura 8.4, la Figura 8.5 y la Figura 8.6, que indica que se comprueban el **28.74 % (1,495 de 5,200)** de las instrucciones de todo el código afectadas por las pruebas, entre muchos otros datos; superando el límite impuesto en los objetivos. Para

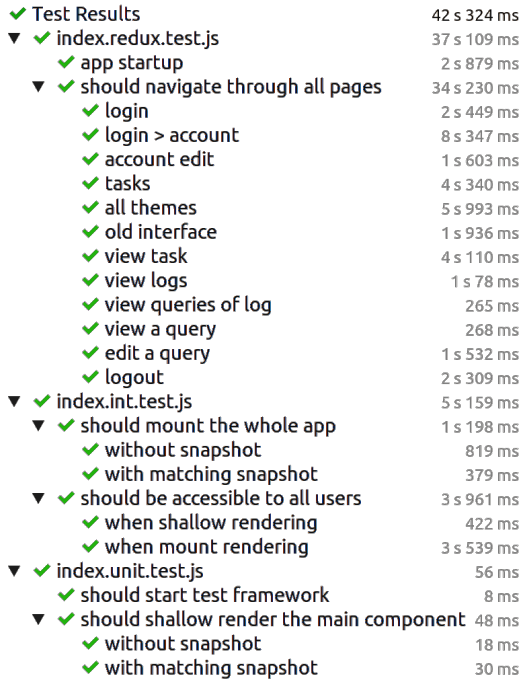
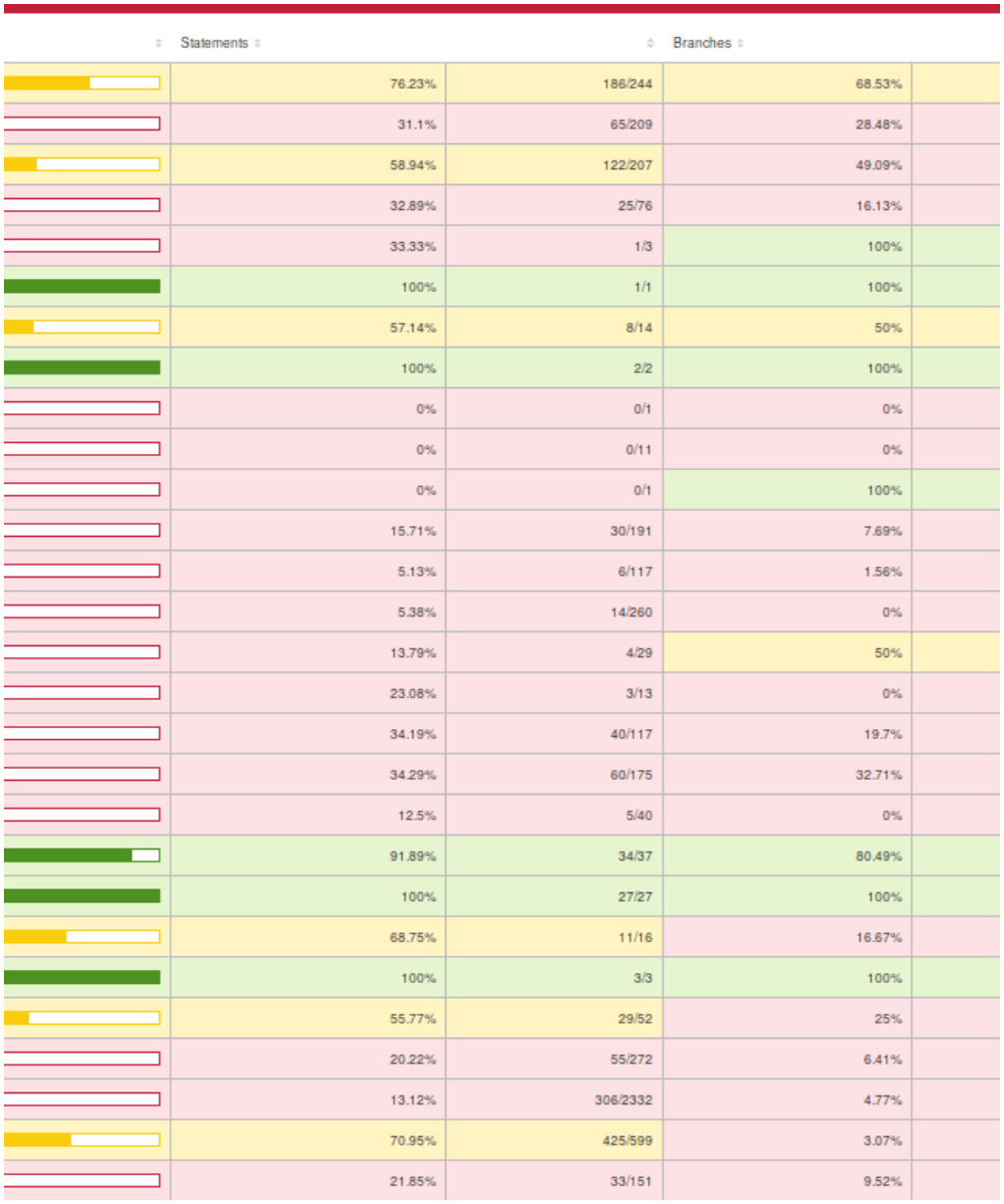


Figura 8.3: Resultados de casos de prueba.

más información se puede descargar el artefacto generado al ejecutar todas las pruebas, el cual incluye la cobertura de los contenidos de los paquetes y también unas vistas de la cobertura de cada clase, en las cuales se muestran las líneas ejecutadas, no ejecutadas y en las que no se ejecutó algún tipo de salto.

Figura 8.4: Cobertura de las pruebas por directorio, parte 1.



	Statements		Branches	
	76.23%	186/244	68.53%	
	31.1%	65/209	28.48%	
	58.94%	122/207	49.09%	
	32.89%	25/76	16.13%	
	33.33%	1/3	100%	
	100%	1/1	100%	
	57.14%	8/14	50%	
	100%	2/2	100%	
	0%	0/1	0%	
	0%	0/11	0%	
	0%	0/1	100%	
	15.71%	30/191	7.69%	
	5.13%	6/117	1.56%	
	5.38%	14/260	0%	
	13.79%	4/29	50%	
	23.08%	3/13	0%	
	34.19%	40/117	19.7%	
	34.29%	60/175	32.71%	
	12.5%	5/40	0%	
	91.89%	34/37	80.49%	
	100%	27/27	100%	
	68.75%	11/16	16.67%	
	100%	3/3	100%	
	55.77%	29/52	25%	
	20.22%	55/272	6.41%	
	13.12%	306/2332	4.77%	
	70.95%	425/599	3.07%	
	21.85%	33/151	9.52%	

Figura 8.5: Cobertura de las pruebas por directorio, parte 2.

	Functions		Lines	
98/143	65.82%	52/79	78.47%	164/209
47/165	21.18%	18/85	31.96%	62/194
54/110	43.96%	40/91	59.68%	111/186
5/31	14.81%	4/27	30.99%	22/71
0/0	0%	0/2	33.33%	1/3
0/0	100%	0/0	100%	1/1
4/8	28.57%	2/7	53.85%	7/13
0/0	100%	0/0	100%	2/2
0/14	0%	0/1	0%	0/1
0/15	0%	0/7	0%	0/11
0/0	0%	0/1	0%	0/1
4/52	14.29%	7/49	15.85%	29/183
1/64	0%	0/32	5.45%	6/110
0/101	0%	0/71	5.83%	14/240
1/2	0%	0/13	15.38%	4/26
0/3	0%	0/8	25%	3/12
13/66	22.58%	7/31	35.51%	38/107
35/107	34%	17/50	33.72%	58/172
0/4	14.29%	2/14	13.16%	5/38
33/41	81.25%	13/16	100%	24/24
4/4	100%	2/2	100%	27/27
1/6	100%	3/3	68.75%	11/16
0/0	100%	0/0	100%	3/3
1/4	23.08%	6/26	60.47%	26/43
10/156	15.07%	11/73	25.94%	55/212
87/1825	8.9%	52/584	15.42%	304/1972
13/424	12%	15/125	73.06%	423/579
8/84	28.21%	11/39	20%	29/145

Figura 8.6: Cobertura de las pruebas por directorio, parte 3.

Capítulo 9

Conclusiones

Se han cumplido todos los objetivos del TFG realizando pruebas de los incrementos analizados, diseñados e implementados, manteniendo una gestión del proyecto correcta usando la metodología por incrementos.

En concreto, se ha implementado el algoritmo base con una aproximación basada en técnicas de conformidad de procesos y en tecnologías de análisis masivo de datos (*Big Data*) y todas las funcionalidades especificadas. Sobre este se añadió procesamiento en paralelo y en un clúster gracias a la adaptación para su uso con *Spark*, permitiendo también leer los datos de una base de datos NoSQL distribuida y consiguiendo limitar el tamaño de los registros con los que trabaja unicamente al almacenamiento disponible en la base de datos, en lugar de la memoria del ordenador del servicio.

Otra ampliación al algoritmo destacada fue el uso de un lenguaje para describir el submodelo de la consulta fácilmente. Se añadió la posibilidad de definir restricciones en *Java*. Se implementó otro lenguaje para definir restricciones que se aplicarán a medida que se ejecuta el algoritmo, apoyándose en los tiempos de inicio y fin, el ciclo de vida y las *KPIs* asociadas a trazas, eventos o globales. Se implementaron distintos modos de ejecución del algoritmo que aumentan la flexibilidad del mismo. El más avanzado de ellos permite cambiar los costes de cualquier paso del algoritmo (aunque sea necesario un salto en el modelo se podría permitir en este modo); descritos mediante el lenguaje de restricciones y definir un *fitness* mínimo para aceptar trazas que se parezcan al menos eso a la consulta en función de su coste. También se desarrollaron extensiones que permiten recuperar y agregar los identificadores de trazas que aceptó la consulta, obtener tiempos medios de tarea y entre tareas y contar tareas.

Se crearon adaptadores para otras bibliotecas usadas en el CiTIUS: inVerbis y otro modelo común a varias bibliotecas. El modelo que usa el algoritmo está basado en el de ProDiGen, ampliado para trabajar con KPIs, tiempos de inicio y fin y ciclos de vida, además de añadir exportación en CSV y mejorar los formatos existentes.

Sobre todo esto se construyó un servidor, que permite el acceso remoto a todas las funcionalidades del incremento anterior, trabajando con usuarios, registros, consultas, restricciones, configuración, ejecución, tareas y resultados del algoritmo. Por último se implementó una interfaz, que permite que un usuario use el algoritmo de forma sencilla y visual.

El algoritmo fue probado extensivamente consiguiendo una cobertura de instrucciones del 64 % y superando todas las pruebas, se analizó su código estáticamente para mejorar su calidad y se realizaron pruebas de rendimiento para comparar las distintas tecnologías disponibles, escogiendo la óptima. Este también se probó de forma satisfactoria con una cobertura de instrucciones del 31 %, ya que este ya se probaría a medida que se desarrolla la interfaz. Para la interfaz se crearon pruebas que fueron superadas con una cobertura de instrucciones del 28.74 % incluyendo una prueba de usabilidad a través de un cuestionario SUS con puntuación de 9.5/10, y pruebas de accesibilidad mediante la biblioteca *axe* superadas.

Este proyecto me permitió aprender muchas tecnologías que no había probado antes, como *Spark*, *Kotlin*, *TypeScript*, *ANTLR*, *Redux*, *JaCoCo*, *Jest*, *Material-UI*..., herramientas de diseño como *yFiles*, *PlantUML*, *JSON Schema*, *GraphQL*, *GraphQL Voyager*... Además amplió mis conocimientos en tecnologías ya probadas durante la carrera, como *Swagger/OpenAPI*, *JUnit 5*, *React*...

9.1. Ampliaciones

Este TFG se ha completado cumpliendo todos los objetivos, pero aún así existen posibles ampliaciones que podrían realizarse en trabajos futuros. Esta tarea no se incluyeron en el TFG para mantener la 420 horas de dedicación indicadas en el anteproyecto.

Añadir *listeners* al algoritmo. Añadiendo más receptores de progreso del algoritmo se pueden realizar más tareas de todo tipo, pero por ejemplo se podría realizar la recolección de otras estadísticas como poder seleccionar entre máximo, mínimo, mediana en lugar de siempre obtener la media de los datos o permitir filtrar algunos pasos del algoritmo no interesantes para la estadística.

Mejorar el lenguaje de consultas. El lenguaje de consultas podría resultar más cómodo si los bucles aceptaran varias tareas actuando cómo secuencia, es decir, convertir estructuras que actualmente son $LOOP(SEQ(...))$ en $LOOP(...)$ para simplificar la lectura.

Mejorar el lenguaje de restricciones. El lenguaje de restricciones puede ser ampliado para ser Turing completo (al igual que lo es el lenguaje de restricciones secundario, que usa *JavaScript*) añadiendo bucles pero manteniendo la simplicidad del lenguaje específico (no como el lenguaje secundario). Este lenguaje también se puede optimizar, por ejemplo, resolviendo operaciones estáticas durante la compilación previa que se realiza (3+2>4 ->TRUE), mejorando el rendimiento de todo el algoritmo.

Mejorar la usabilidad de la interfaz. La interfaz recibió una puntuación de 9.5 sobre 10, pero aún así se puede mejorar comprobando los ítems del cuestionario SUS que redujeron la puntuación.

Mejorar los grafos de consulta. Una mejora sería mostrar los tiempos y número de ocurrencias de los arcos en el grafo al mostrar los resultados. Con la implementación actual, se disponen de los datos y se muestran datos de ejemplo porque no se conectaron aún los generadores de grafos a los resultados.

Mejorar las tablas de búsqueda. Se pueden añadir aún más formas de búsqueda en todas las tablas, permitiendo filtrar por igualdad o si cumplen una expresión regular además de la forma de búsqueda por defecto que consiste en aceptar los elementos que contengan la cadena de texto. El servidor ya permite estas búsquedas, pero es una funcionalidad que no se llevó hasta la interfaz.

Mejorar el editor de la interfaz. También se podría mejorar el editor en modo texto añadiendo facilidades para la predicción. La predicción ya es posible con la configuración actual, y se puede ver una parte de la misma en los mensajes de error, que sugieren como continuar a partir del error; pero se podría implementar autocompletado de elementos en la interfaz, lo cual incluiría símbolos como operaciones e incluso elementos variables entre registros como eventos (ya se dispone de la lista de eventos y se muestra sobre el editor en modo texto). Esta característica no se implementó porque existe una muy similar en el editor en modo árbol que ya filtra las acciones si no están disponibles, y porque implicaría superar las horas asignadas al TFG.

Apéndice A

Manuales técnicos

Para tener una visión general de las tareas principales a realizar sobre cada resultado de incremento (como compilar, probar, obtener análisis de cobertura, ejecutar o desplegar), se puede ver el fichero de configuración de integración continua en el Apéndice D, que muestra cómo se realiza cada una de estas tareas sobre un sistema base usando imágenes de Docker y series de comandos de forma funcional, probada en cada subida de código al repositorio (CI) y replicable de forma sencilla en otro sistema.

A.1. Manual de uso como biblioteca

La biblioteca compilada está disponible en el repositorio de Nexus mantenido en el CiTIUS siempre actualizada a la última versión disponible, y con alguna versión anterior. De todas formas realizar la instalación manual es muy fácil:

1. Desde conformancechecking/conformancechecking/: `mvn install -DskipTests`

Los muchos test creados para el proyecto del algoritmo pueden resultar útiles para ver el uso del algoritmo como biblioteca. En general, los pasos serían:

1. Cargar y/o adaptar el log (generar un `ILog`). A partir de este se obtiene el `KPIProvider` (`KPIProviderFromProDiGenILog`) necesario si se usan restricciones. Si se desea usar Spark [10], debería generarse un log de tipo `MLogSpark`.

2. Cargar y/o adaptar la consulta (generar un `CMIndividual`)

3. Instanciar `Alignments`. Se puede usar el constructor más simple para mantener los argumentos por defecto hasta que se necesiten.

4. Opcionalmente introducir restricciones con `alignments.setRestrictions()` y de argumento algo que implemente la interfaz `IRestriction`, como el lenguaje `CustomConditionsLanguage` que se carga a partir de texto.

5. Llamar a `alignments.execute(ILog)`.

El conjunto de tests de `TestCaseAmtega.java` es bastante completo, trabajando con restricciones y consultas, por lo que es un buen punto de comienzo para integrar con otros registros, consultas...

La clase `LogLocalAdapterInVerbis` es la que más se usará para realizar la conversión de los registros para la empresa interesada inVerbis. Además esa clase ofrece información de progreso, y se puede envolver con almacenamiento o caches automáticas con las clases de adapters/stored y adapters/cached para realizar conversiones posteriores instantáneas.

Para más información se puede consultar la documentación disponible en el código con comentarios Javadoc.

A.2. Manual de instalación de la aplicación

A.2.1. Compilar el JAR

La biblioteca compilada está disponible en el repositorio de Nexus mantenido en el CiTIUS siempre actualizada a la última versión disponible, y con alguna versión anterior. De todas formas realizar la instalación manual es muy fácil:

1. Si la compilación del algoritmo no estuviera actualizada, seguir el paso de la Sección A.1: Manual de uso como biblioteca.

2. Si la compilación de la interfaz no estuviera actualizada, sería suficiente con ejecutar `npm build` desde la carpeta de la misma.

3. Desde conformancechecking/conformancecheckingbackend/:
   `./gradlew bootServerJar`

4. (Opcional si se va a usar un cluster) Desde conformancechecking/conformancecheckingbackend/:
   `./gradlew bootClusterJar`

A.2.2. Ejecutar en local

Preparar dependencias

MongoDB Para el correcto funcionamiento del servicio (que usa transacciones para asegurar la atomicidad de las operaciones), es necesario una versión >= 4.0 ejecutada con la opción –replSet "something" y en la que se ha ejecutado rs.initiate(). Además se debe ejecutar db.adminCommand(setFeatureCompatibilityVersion: "'4.0") para soportar transacciones a través de sesiones y proporcionar un entorno más seguro.

Cuando se llege a la Sección A.2.2: Configurar, se puede indicar el host, el puerto y la base de datos de mongo que se usarán, aunque probablemente no sea necesario cambiarlos. Esto se hace en el fichero *application.yaml*:

```
mongo:
  host: "localhost"
  port: 27017
  remoteHost: "localhost"
  remotePort: 27017
  localDatabaseName: "conf_chk_local"
  remoteDatabaseName: "conf_chk_remote"
```

Configurar

La configuración para la ejecución en local es opcional, pero proporciona muchas utilidades como cambiar el puerto del servidor, cambiar la configuración HTTPS (por ejemplo cambiar la clave o desactivar HTTP/2), cambiar el tiempo de cuentas de invitado o desactivarlas, etc.

Para configurar se crea o edita el fichero application.yml y application.properties, dentro o fuera del JAR en la misma carpeta y las claves y comentarios ya indican lo que configura cada opción.

Ejecutar

1. `java -jar <ejecutable>.jar`

A.2.3. Ejecutar en remoto

Preparar dependencias

MongoDB Ver Sección A.2.2: MongoDB. Se recomienda configurarlo tanto en el servidor como el clúster para aprovechar la localidad de los datos y mejorar la eficiencia del algoritmo. Se recomienda revisar la configuración de ambos ejecutables para que apunten a las instancias correctas de Mongo, manteniendo los metadatos cercanos al servicio y los registros y consultas en sí cercanos al clúster que trabajará sobre ellos.

Configurar

Ver Sección A.2.2: Configurar. Es necesario configurar la ubicación del clúster remoto. Además, como el clúster remoto debe ser accesible desde el servidor y suele ser relativamente público, es recomendable cambiar la contraseña para que sólo el servicio configurado correctamente sea capaz de comunicarse y ejecutar el algoritmo. La configuración de la que se habla está en *application.properties* del servicio:

```
# NO_REMOTE_CLUSTER / http://127.0.0.1:48613/ /
↪ https://127.0.0.1:48613/
remote.cluster=NO_REMOTE_CLUSTER
remote.password=password
```

Ejecutar

En el servicio:

1. `java -jar <ejecutable_servidor>.jar`

En remoto (si se usa un sólo computador):

1. `password=<password> java -jar <ejecutable_cluster>.jar`

Para ejecutar en clúster remoto aprovechando las capacidades de *Spark*, el procedimiento sería el estándar, que varía según la instalación del clúster. En el clúster *Big Data* 1 CiTIUS se puede desplegar mediante el comando:

1. `spark-submit -class es.usc.citius.alignments.backend.ApplicationLauncher -master yarn -deploy-mode cluster <ejecutable_cluster>.jar`

A.3. Documentación de los servicios del servidor

Disponible en fichero adjunto de nombre *OpenAPI*.

Apéndice B

Manual de usuario

B.1. Manual de instalación de la aplicación

Ver Sección A.2: Manual de instalación de la aplicación.

B.2. Manual de uso de la interfaz

B.2.1. Acceso a la interfaz

La interfaz optimizada está integrada en el servidor, el cual la sirve si se accede a https://<host>:<puerto>/, donde el puerto por defecto es 8080. También se puede desplegar la interfaz por si sola usando `npm serve` en el código de la misma, en cuyo caso también estaría disponible la versión de desarrollo en la página raíz del servidor iniciado, que por defecto se inicia en el puerto 3000.

B.2.2. Elementos comunes de la interfaz

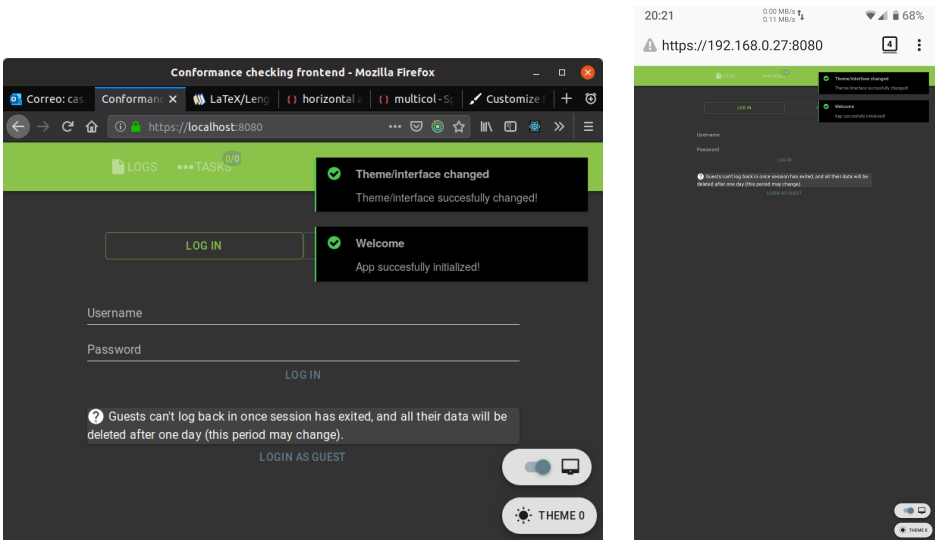


Tabla B.1: Elementos comunes de la interfaz en ordenador y móvil.

En la Tabla B.1 se muestra la pantalla de inicio de sesión completa para explicar una sola vez los elementos reusados en toda la interfaz.

Se usa la URL para indicar la página actual en la que se encuentra el usuario a pesar de que se trata de una *Single-Page Application*. Esta se recuerda entre sesiones permitiendo que, por ejemplo, si un usuario estaba trabajando en una consulta cuando inicie sesión otra vez vuelva a la misma consulta, con acceso rápido mediante la barra superior a otras vistas.

Después, la barra superior permite navegar por las páginas principales: registros (desde los que se accede a consultas, resultado...), tareas y cuenta (tapada por notificaciones). Estos se explicarán en secciones posteriores, pero la pestaña de tareas tiene el añadido de presentar contadores de tareas que se están ejecutando actualmente y las totales registradas.

Además, se pueden ver notificaciones arriba a la derecha que notifican ciertos cambios de estado en la aplicación al usuario además de usarse para una previsualización de las tareas en tiempo real que está ejecutando el servicio (también accesibles en la pestaña tareas). Están animadas, siempre se pueden esconder manualmente haciendo clic sobre ellas o pulsándolas o arrastrándolas en un dispositivo móvil y normalmente se esconden automáticamente a los pocos segundos.

Por último, existen dos botones abajo a la izquierda que manejan la interfaz usada (anterior o posterior al rediseño), y el tema usado (existen 5 temas diseñados para la aplicación aplicables globalmente). Ambos se pueden cambiar en cualquier momento y se guarda la configuración entre reinicios de la interfaz gracias a *Redux*. Destaca que existen temas claros de fondo blanco para los usuarios que lo prefieran.

En futuras secciones se obviarán estos elementos comunes, mostrando sólo la parte central de la página, que es la que varía. Esta se adapta dinámicamente al dispositivo de visualización: por muy ancho que sea, no supera los 1240px, 1024px o 600px dependiendo de la configuración de la vista y se mantiene centrada, reduciéndose si el dispositivo es más pequeño. El móvil indica una alerta de seguridad por haber aceptado un certificado SSL de prueba, pero esto no ocurriría en producción si se dispone de un certificado autorizado.

B.2.3. Registro e inicio de sesión

En la Tabla B.2 se muestran la vistas de inicio de sesión y registro.

El email es opcional, pero el nombre y la contraseña no pueden estar vacíos. Si ocurre algún error durante el inicio de sesión se indica debajo. El error cuando falla el inicio de sesión es “Forbidden”.

Hay dos opciones para crear una cuenta: el registro o el uso de la herra-

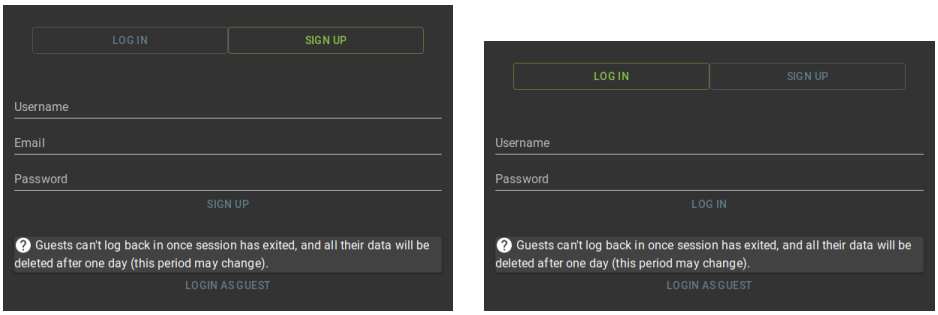


Tabla B.2: Registro e inicio de sesión.

mienta como invitado. La diferencia es que el registro es permanente, mientras que el uso como invitado es instantáneo pero no almacena los datos de forma persistente, sino que toda la cuenta se borra en un tiempo configurado en el servidor (por defecto de un día). Se ofrece la segunda opción para acelerar el uso de la herramienta en ciertos casos de uso que sólo buscan ciertos resultados sin almacenar nada de forma permanente, con limpieza automática por el servidor.

Una vez registrado correctamente un usuario, se cubren automáticamente los campos de inicio de sesión para facilitar el mismo. Lo mismo ocurre cuando se inicia sesión como invitado (en este caso no es necesaria interacción del usuario para iniciar sesión).

B.2.4. Cuenta y edición de cuenta

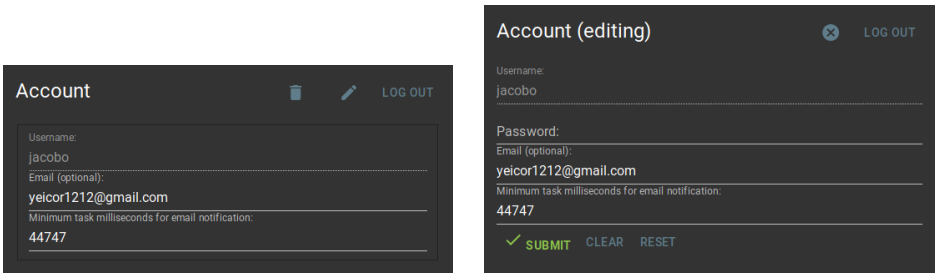


Tabla B.3: Cuenta y edición de cuenta.

En la Tabla B.3 se muestran la vistas de la cuenta y edición de la cuenta. Se accede a la segunda vista pulsando el lápiz de la primera, y se vuelve a la primera pulsando la x en la segunda.

Estas vistas permiten ver y editar la configuración del usuario; se puede editar todo excepto el nombre del usuario.

Desde ambas se puede cerrar sesión y desde solo la primera vista se puede borrar la cuenta para evitar posibles confusiones.

El último campo es el tiempo mínimo que debe transcurrir durante la ejecución de una tarea para que se envíe un correo informando de su finalización ya sea correcta o por error. Este correo incluye un enlace a la interfaz para comprobarla, la información de la tarea y el registro completo de la tarea.

B.2.5. Tareas y tarea

Tabla B.4: Tareas y tarea.

En la Tabla B.4 se muestran la vistas de tareas y tarea. La difrencia es que la segunda permite mostrar toda el registro de la tarea en toda la pantalla, mientras que la primera tiene el alto limitado para cada tarea. Se accede a la segunda vista pulsando el botón de la izquierda en la primera y se puede volver a la primera mediante la barra superior.

La acción que se puede realizar sobre las tareas es cancelarlas si están en ejecución, o olvidarlas en la interfaz si ya han terminado. En la primera vista también se pueden eliminar todas las tareas finalizadas para facilitar esta tarea.

B.2.6. Listado registros

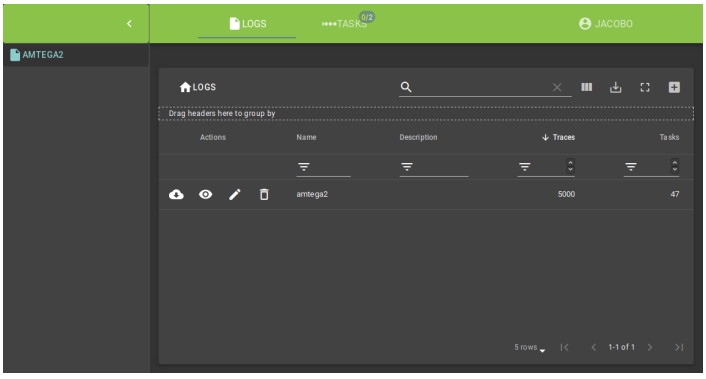


Figura B.1: Registros.

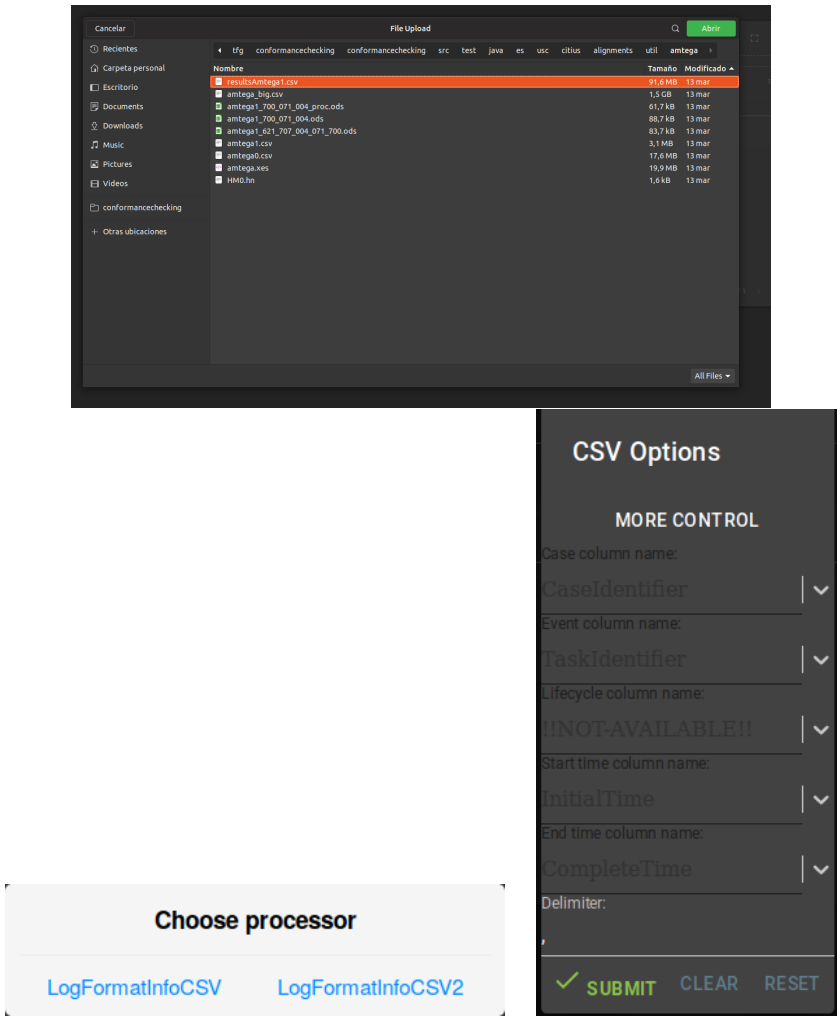


Tabla B.5: Registros: pasos para importar.

En la Figura B.1 se muestra la vista de los registros cargados por el usuario actual. Esta y las derivadas de esta disponen de un menú izquierdo desplegable que por defecto está abierto según el tamaño de la pantalla. Este contiene registros y consultas formando un árbol. No los muestra todos, sino que muestra los que se han consultado previamente para evitar sobrecargar con datos la interfaz.

La tabla central permite realizar muchas acciones para poder encontrar el registro deseado cuando se hayan cargado muchos en la herramienta. Estas incluyen búsqueda (con filtrado, ordenado y agrupación por cualquier columna y cambiar de página y el tamaño de página), añadir un registro (importarlo, introduciendo los metadatos nombre y descripción previamente, ver Tabla B.5), exportar un registro (ver Figura B.2, con la configuración similar a la importación de registros), borrar (pide confirmación), editar metadatos y acceder al registro (ver). La importación de registros en CSV con el primer procesador tie-

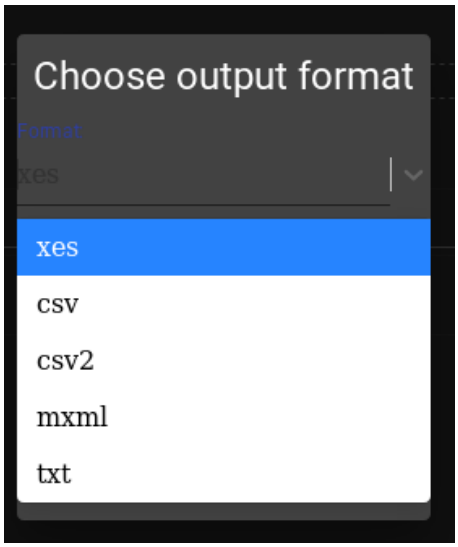


Figura B.2: Registros: exportar.

ne la ayuda de identificar los nombres de columnas automáticamente del fichero para permitir seleccionarlos fácilmente como se ve en la Tabla B.5, mientras que el segundo tiene una mayor eficiencia, reduciendo las opciones de configuración disponibles y sin la capacidad de identificar automáticamente el formato de las fechas, ni de que este varíe.

B.2.7. Listado consultas

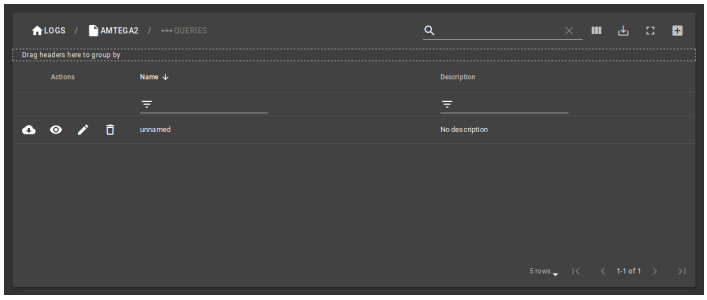


Figura B.3: Consultas.

En la Figura B.3 se muestran la vista de consultas, que contiene el mismo menú lateral desplegable que los registros.

La tabla tiene los mismos controles que la descrita en el apartado anterior, con cambios en la importación y exportación ya que se trata con otros formatos (CM y CN) y se simplifico mucho la interfaz (se trata en apartados siguientes).

Figura B.4: Crear/editar consulta.

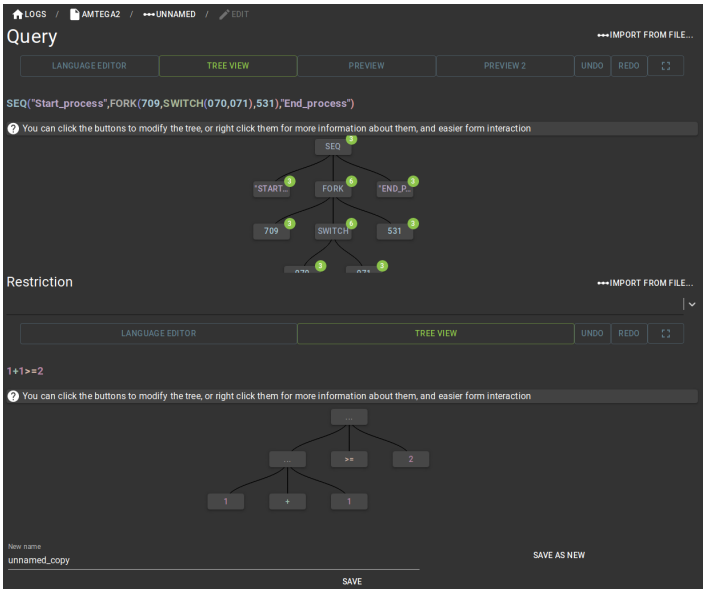


Figura B.5: Crear/editar consulta 2.

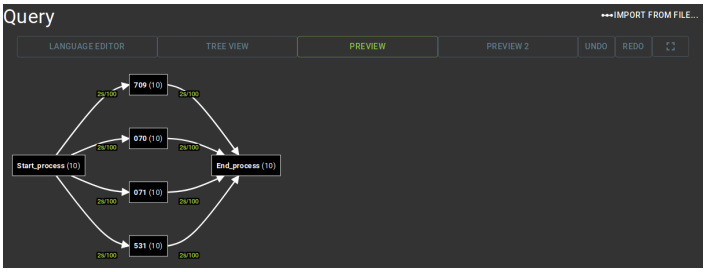


Figura B.6: Crear/editar consulta 3.

B.2.8. Crear/editar/duplicar consulta

En la Figura B.4 se muestran la vista de creación y edición de consultas, que contiene el mismo menú lateral desplegable que los registros. En Figura B.5, Figura B.6 y Figura B.7 se muestran las distintas partes de ambos editores usados. Estos tienen controles genéricos de deshacer, rehacer y ver en pantalla

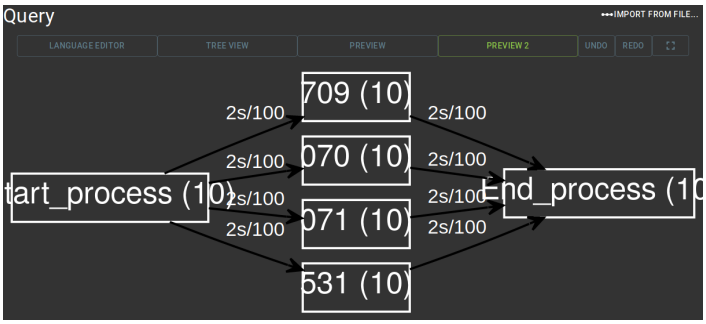


Figura B.7: Crear/editar consulta 4.

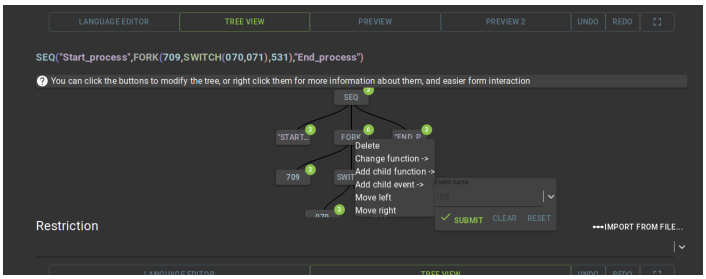


Figura B.8: Crear/editar consulta 5.

Figura B.9: Crear/editar consulta 6.

completa.

Esta es la vista avanzada que permite diseñar consultas mediante sus modelos y sus restricciones a aplicar. Ambas se manejan con editores de lenguajes similares. Estos contienen el editor en texto que analiza sintácticamente los contenidos coloreándolos según el lexema, y se incluyen las actividades disponibles en el registro para facilitar la composición de estos lenguajes. En caso de error sintáctico se emite un mensaje debajo del editor para facilitar la corrección del error mediante una descripción del mismo, además de marcar en el texto el punto o los puntos de error.

En la segunda pestaña del editor se muestra una visualización en árbol (Figura B.5, Figura B.8 y Figura B.9), que en el caso del lenguaje del modelo, permite la edición mediante acciones aplicables sobre los nodos del mismo, como

añadir un hijo al nodo (un argumento a la función), editar una función o un evento, borrar... Estos se representan de dos formas: mediante clic izquierdo con un menú contextual, y con clic derecho con un menú avanzado similar a las tablas anteriores.

Las últimas pestañas, solo presentes en el lenguaje del modelo, permiten construir una previsualización del grafo únicamente usando la interfaz, sin necesidad de estar conectado al servicio. Esto permite a los usuarios verificar que están construyendo la consulta que quieren. Se recomienda usar la primera previsualización porque consigue un mayor rendimiento por la biblioteca usada, y está mejor diseñada.

Esta vista también permite importar tanto modelos como restricciones desde fichero de exportaciones previas, lo cual se hace pulsando los botones "import from file...", los cuales abren un menú para seleccionar el fichero similar al explicado para registros, y si fuera necesario también ofrecerían configuración.

Los botones para confirmar la consulta dependen del estado: creando/editando/editando una consulta que ya tiene resultados. En el primer caso se puede crear unicamente. En el segundo se puede guardar, sobreescribiendo la consulta anterior, o guardar como una copia con el nombre introducido al lado para mantener ambas versiones disponibles. En el último caso sólo se puede guardar como copia.

B.2.9. Ver consulta, ejecutar y resultados

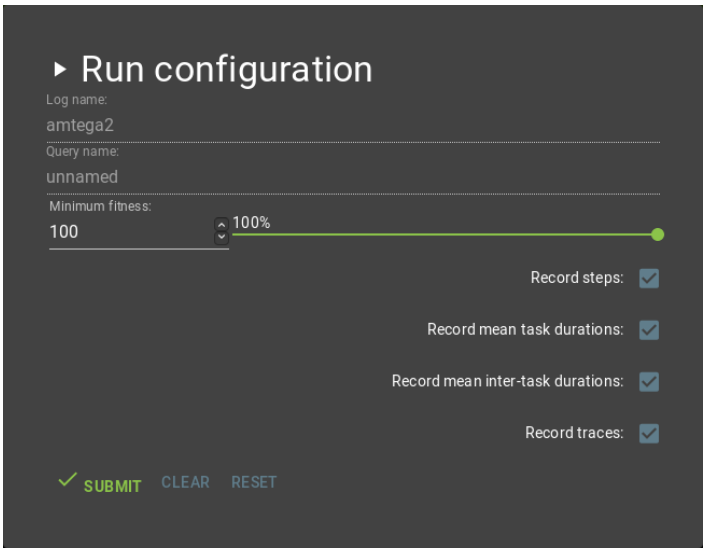


Figura B.10: Configurar algoritmo.

En la Figura B.12 se muestran la vista de la consulta con sus restricciones y parte de los resultados. En este ejemplo no hay restricciones, pero se verían de

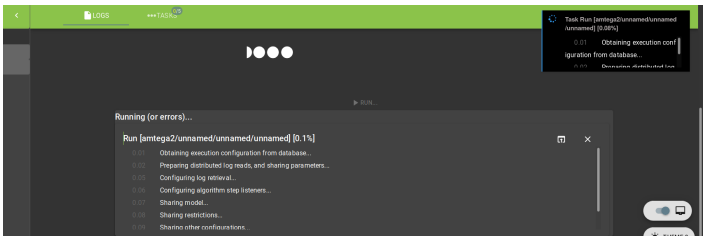


Figura B.11: Tarea ejecución.

forma similar al editor sin capacidad de editar y con la visualización de árbol por defecto.

En la parte superior se ven la consulta y la restricción, sin la capacidad de editarlas. Se dispone de un elemento superior de ayuda a la localización que permite saltos rápidos al registro y a la lista de registros. Este componente también está presente en muchas otras vistas para facilitar la navegación. Arriba a la derecha hay un botón que permite editar la consulta, descrito en otro apartado. Además existen botones para exportar tanto la consulta como la restricción.

A continuación está el botón de ejecución, que estaría activado si no se ha ejecutado antes esta consulta (siempre es posible duplicar la consulta desde el menú de edición para volver a ejecutarla). Este botón muestra la configuración del algoritmo, que el usuario puede editar a su gusto para esta consulta. Esta se muestra en la Figura B.10.

Una vez configurado empieza a ejecutarse, lo que genera una tarea. Esta tarea, además de tratarse como una tarea normal (vista de tareas y notificación), se muestra en la propia página de visualización de la consulta, con el registro en tiempo real. El usuario puede comprobar fácilmente el progreso desde esta misma página. Esto se muestra en la Figura B.11. En esta también se ve cómo se está cargando un componente de forma asíncrona, mientras que al mismo tiempo se muestran por pantalla algunos componentes ya cargados, mejorando la experiencia de usuario.

Por último, tras la ejecución, se pueden ver la configuración utilizada y los resultados en la parte final de la vista. Existen dos opciones sobre estos resultados: exportarlos como fichero JSON para ser postprocesado, o guardar las trazas válidas como un nuevo registro (si se activó la opción de guardar trazas válidas). De esta forma se pueden realizar nuevas consultas sobre el conjunto de trazas que fueron aceptadas por las consultas previas.

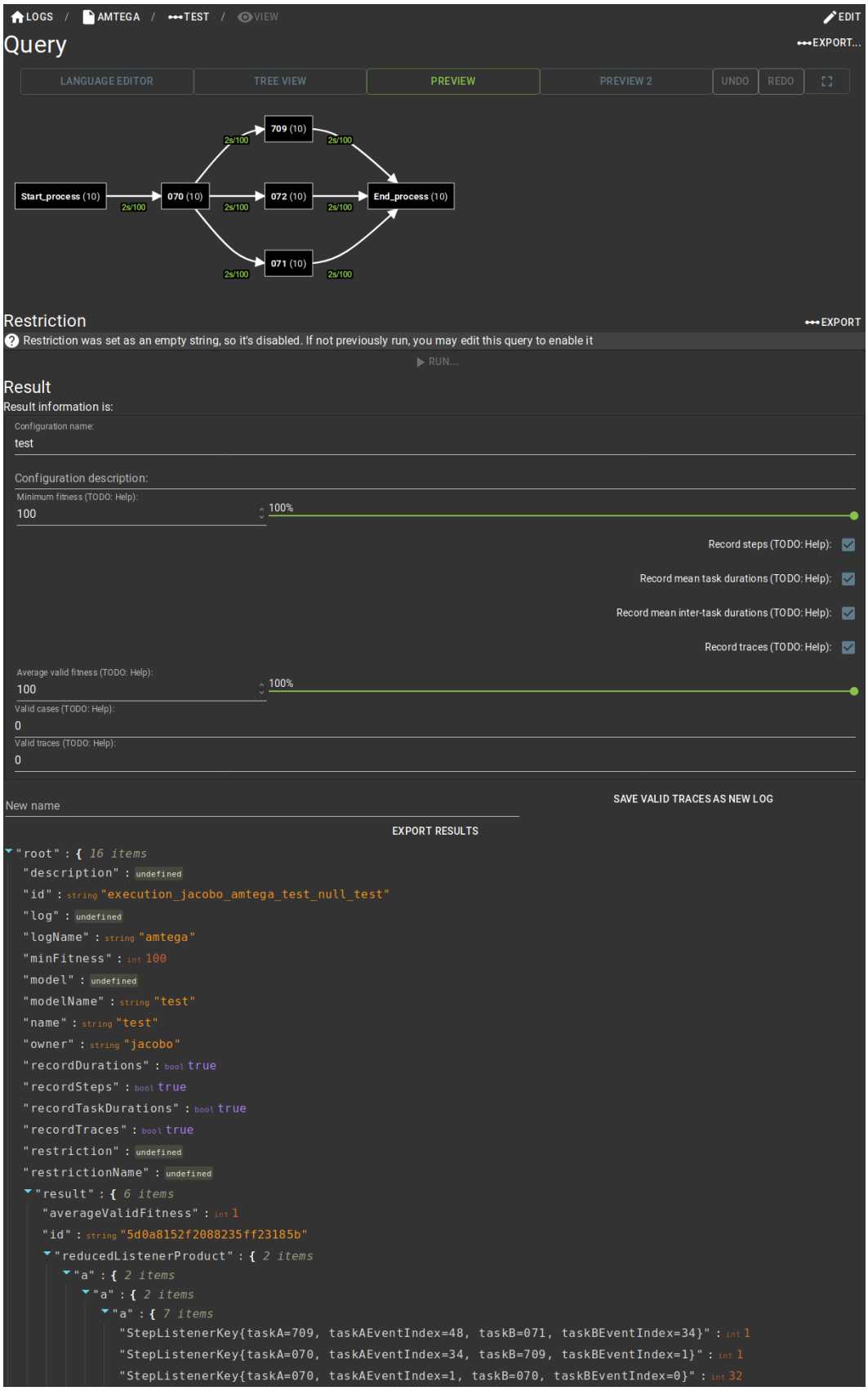
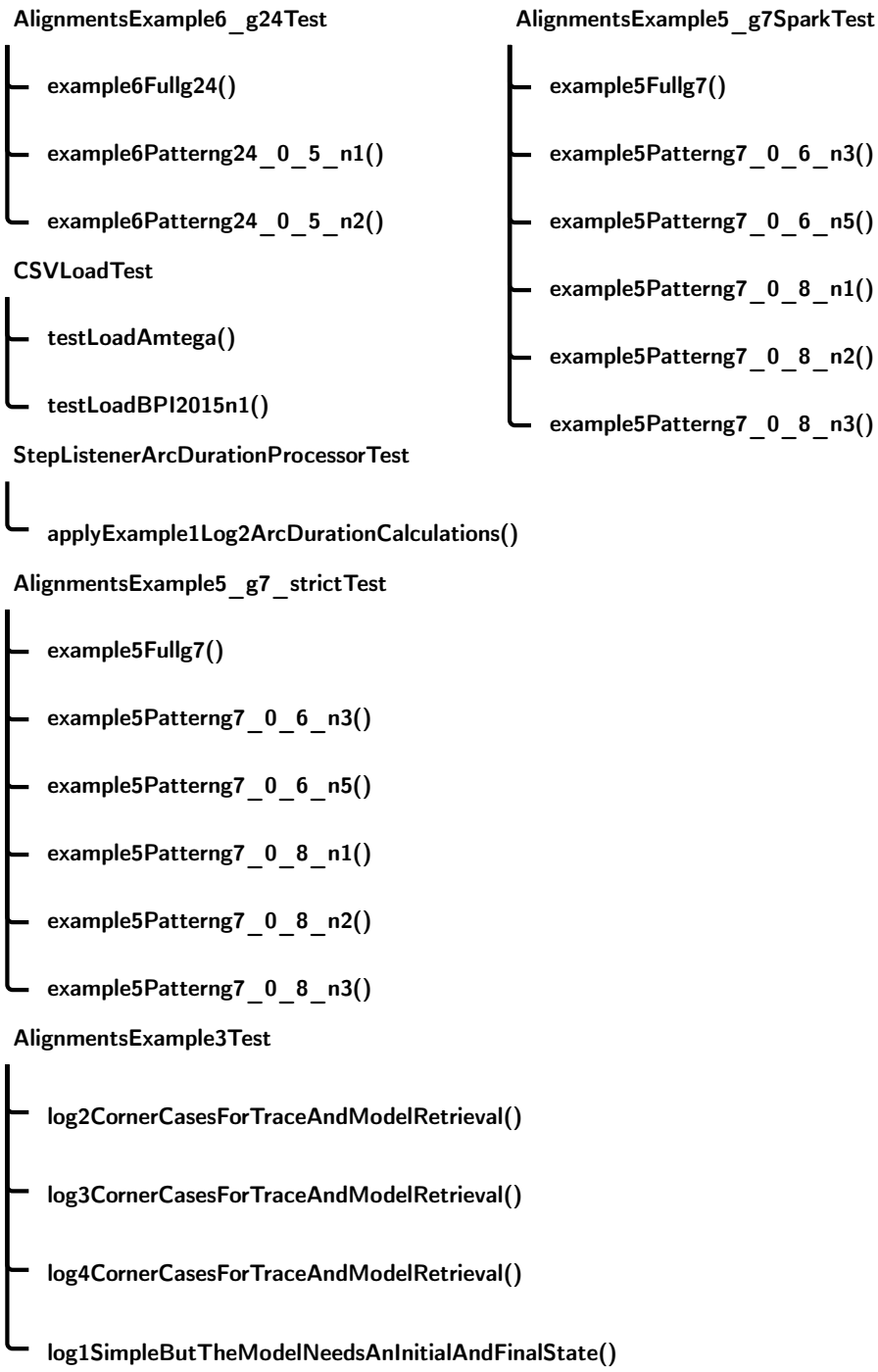


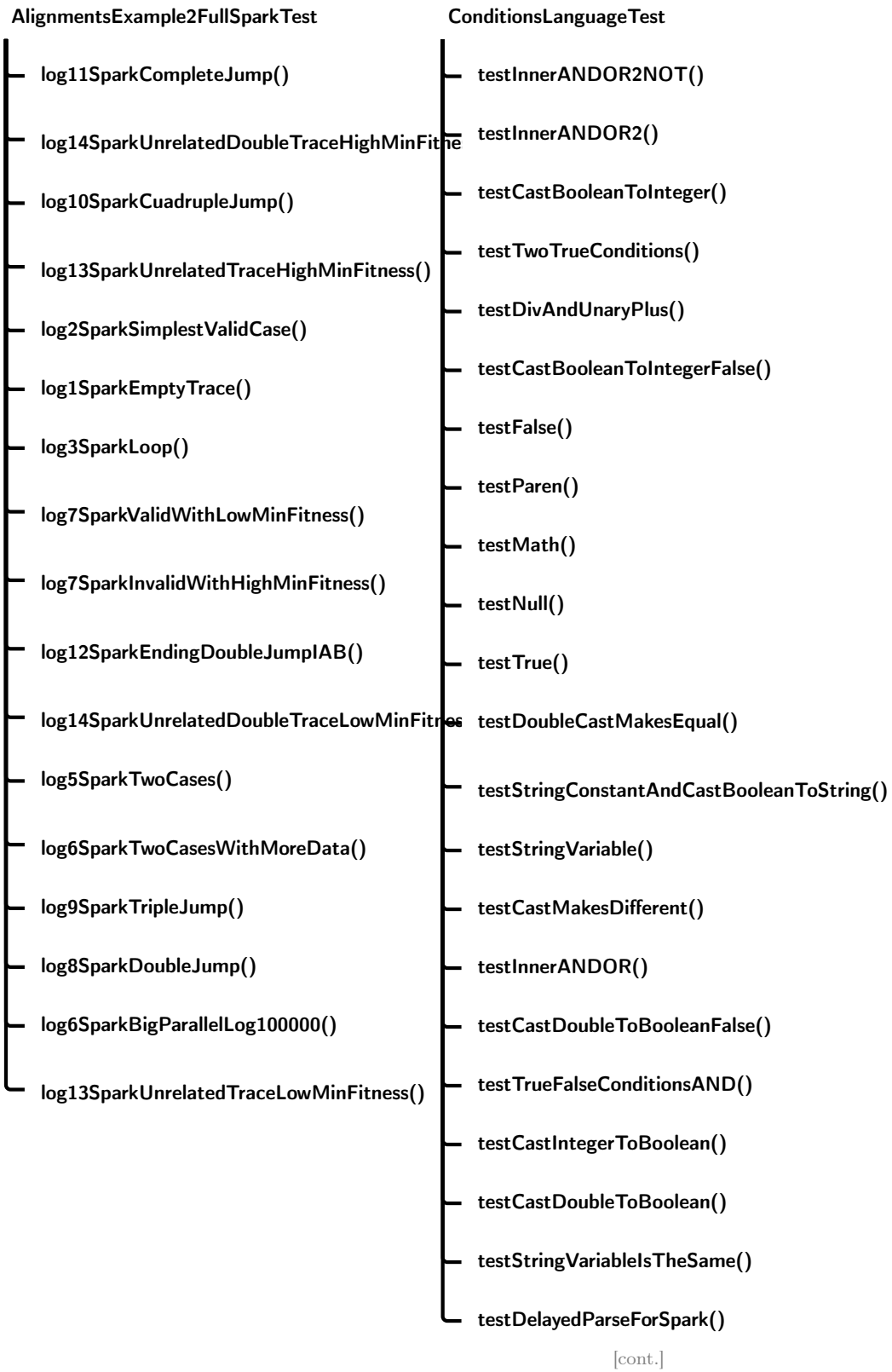
Figura B.12: Ver consulta y resultados.

Apéndice C

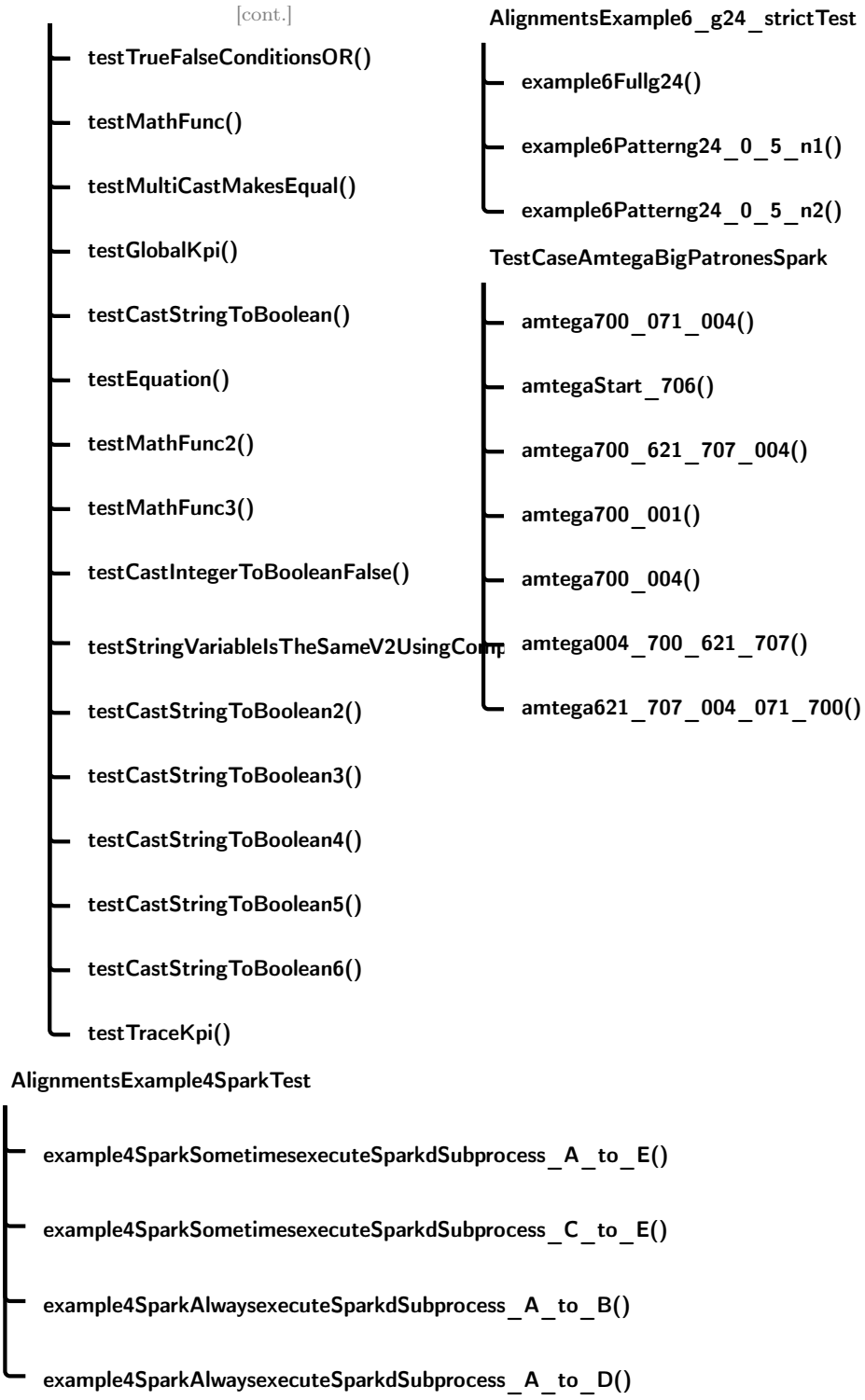
Incremento 1: Casos de prueba

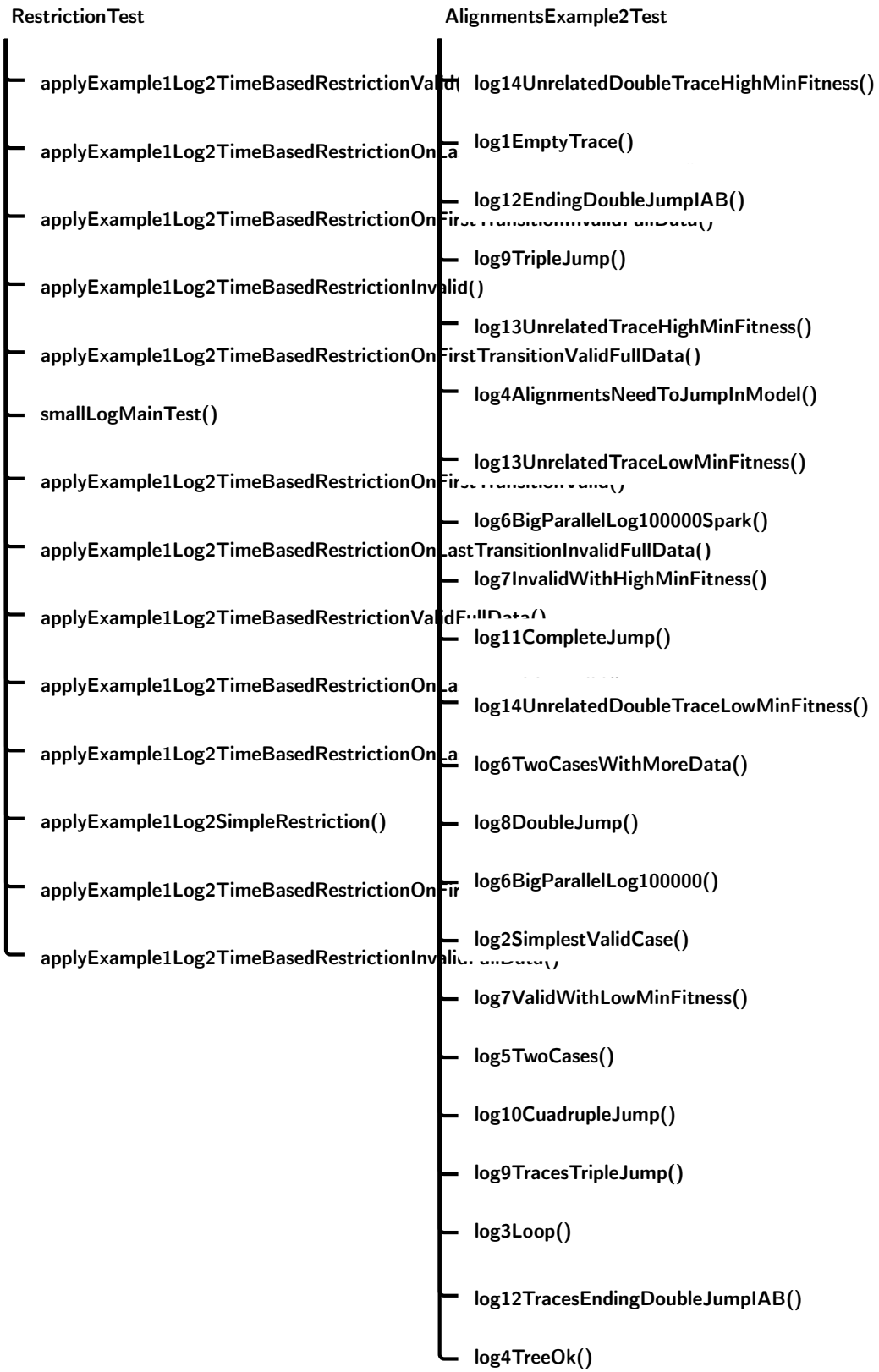
A continuación se indican los casos de prueba desarrollador para cada prueba ya explicada en la Tabla 5.4.

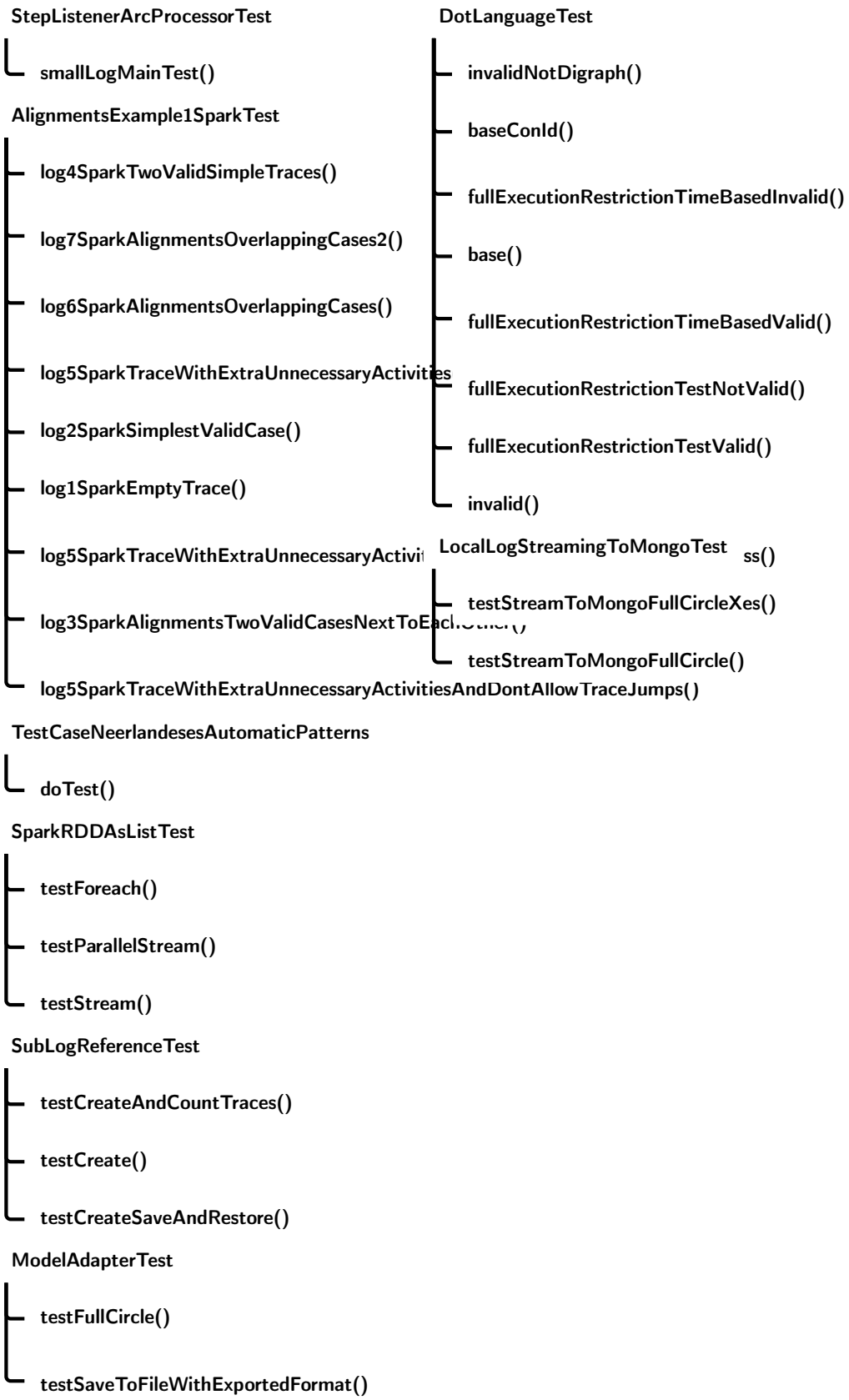

AlignmentsExample6_g24Test
example6Fullg24()
example6Patterng24_0_5_n1()
example6Patterng24_0_5_n2()
CSVLoadTest
testLoadAmtega()
testLoadBPI2015n1()
StepListenerArcDurationProcessorTest
applyExample1Log2ArcDurationCalculations()
AlignmentsExample5_g7_strictTest
example5Fullg7()
example5Patterng7_0_6_n3()
example5Patterng7_0_6_n5()
example5Patterng7_0_8_n1()
example5Patterng7_0_8_n2()
example5Patterng7_0_8_n3()
AlignmentsExample3Test
log2CornerCasesForTraceAndModelRetrieval()
log3CornerCasesForTraceAndModelRetrieval()
log4CornerCasesForTraceAndModelRetrieval()
log1SimpleButTheModelNeedsAnInitialAndFinalState()
AlignmentsExample5_g7SparkTest
example5Fullg7()
example5Patterng7_0_6_n3()
example5Patterng7_0_6_n5()
example5Patterng7_0_8_n1()
example5Patterng7_0_8_n2()
example5Patterng7_0_8_n3()

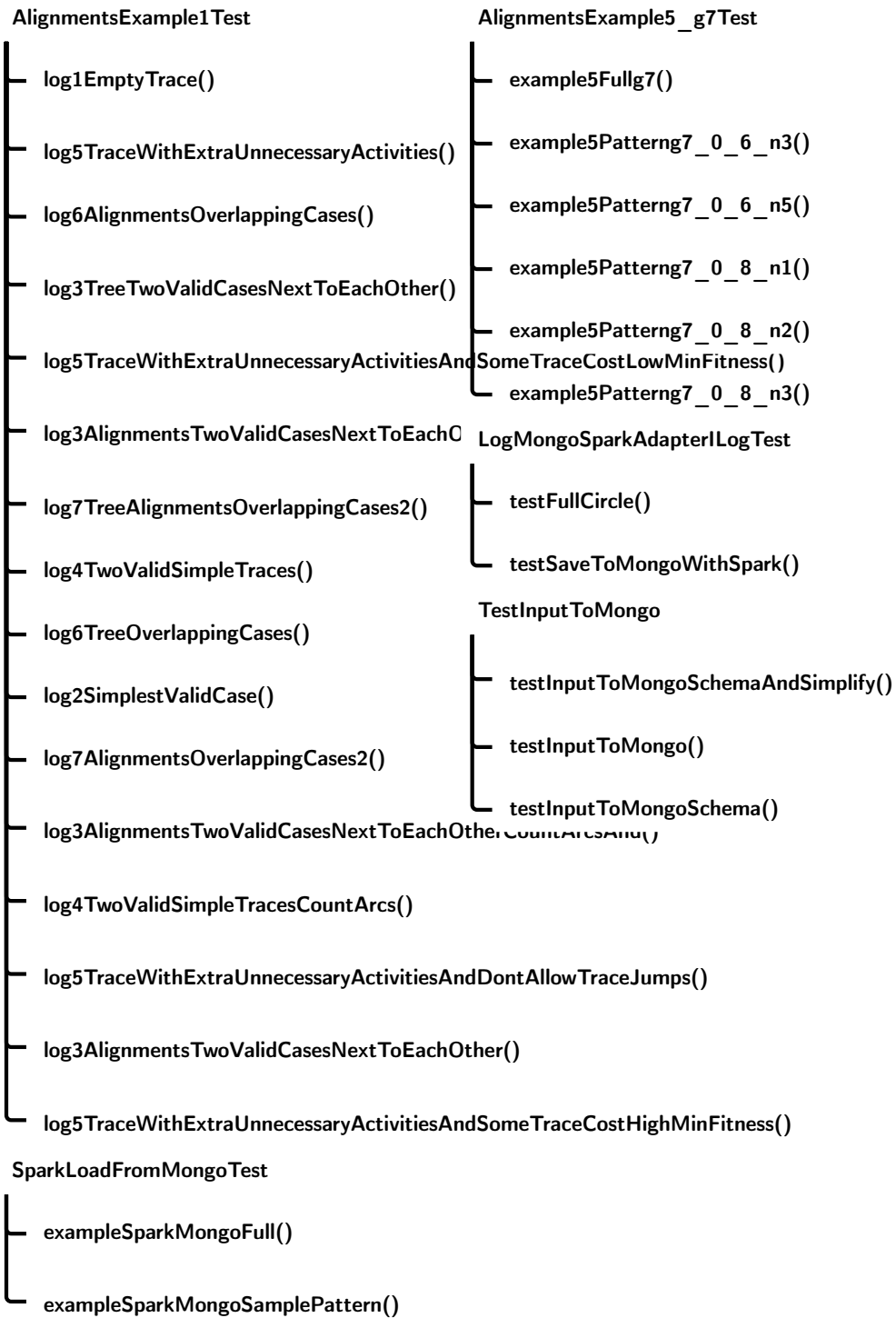

AlignmentsExample2FullSparkTest
log11SparkCompleteJump()
log14SparkUnrelatedDoubleTraceHighMinFitne
log10SparkCuadrupleJump()
log13SparkUnrelatedTraceHighMinFitness()
log2SparkSimplestValidCase()
log1SparkEmptyTrace()
log3SparkLoop()
log7SparkValidWithLowMinFitness()
log7SparkInvalidWithHighMinFitness()
log12SparkEndingDoubleJumpIAB()
log14SparkUnrelatedDoubleTraceLowMinFitnes
log5SparkTwoCases()
log6SparkTwoCasesWithMoreData()
log9SparkTripleJump()
log8SparkDoubleJump()
log6SparkBigParallelLog100000()
log13SparkUnrelatedTraceLowMinFitness()
ConditionsLanguageTest
testInnerANDOR2NOT()
testInnerANDOR2()
testCastBooleanToInteger()
testTwoTrueConditions()
testDivAndUnaryPlus()
testCastBooleanToIntegerFalse()
testFalse()
testParen()
testMath()
testNull()
testTrue()
testDoubleCastMakesEqual()
testStringConstantAndCastBooleanToString()
testStringVariable()
testCastMakesDifferent()
testInnerANDOR()
testCastDoubleToBooleanFalse()
testTrueFalseConditionsAND()
testCastIntegerToBoolean()
testCastDoubleToBoolean()
testStringVariableIsTheSame()
testDelayedParseForSpark()
[cont.]

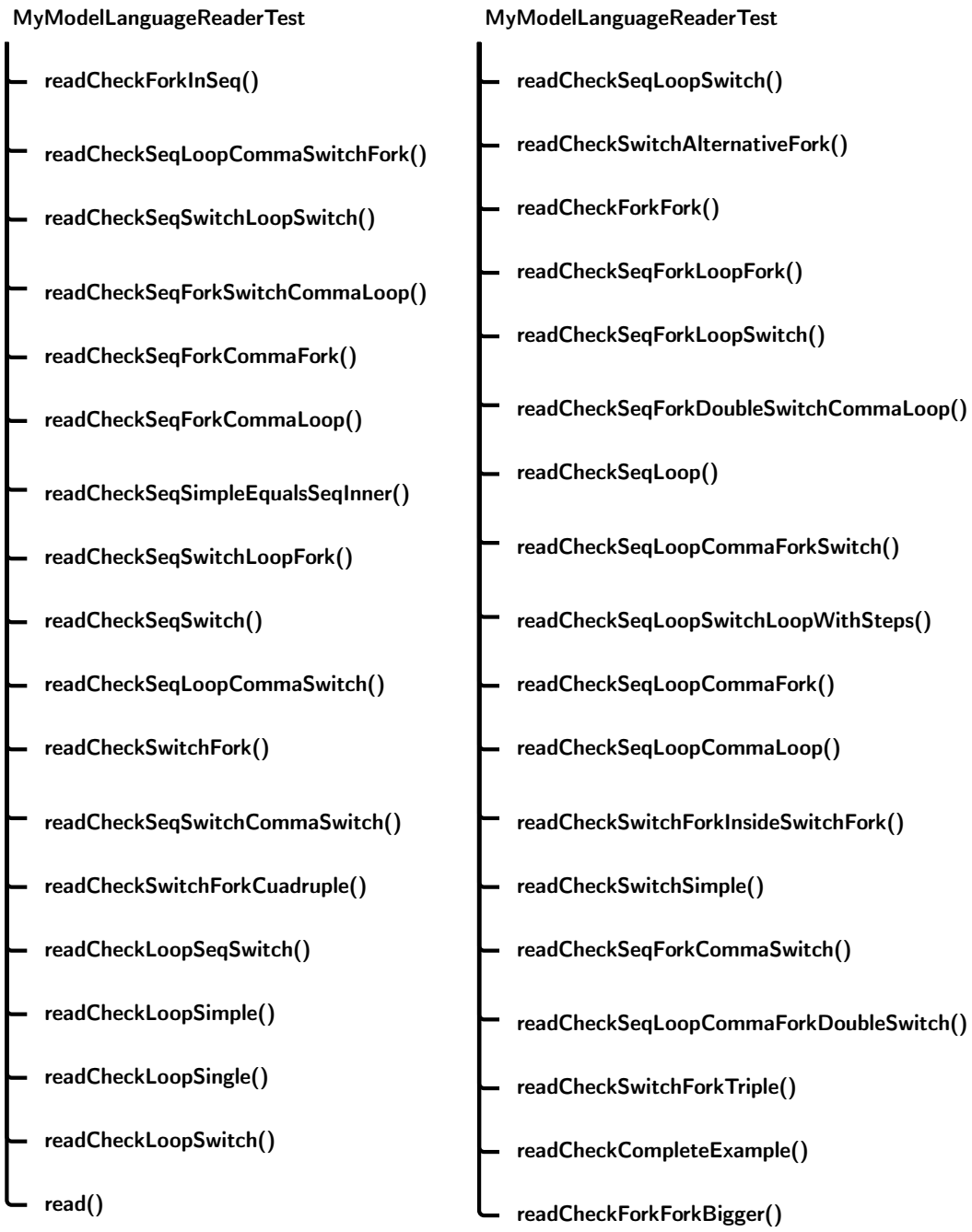
continuado


[cont.]
testTrueFalseConditionsOR()
testMathFunc()
testMultiCastMakesEqual()
testGlobalKpi()
testCastStringToBoolean()
testEquation()
testMathFunc2()
testMathFunc3()
testCastIntegerToBooleanFalse()
testStringVariableIsTheSameV2UsingCom
testCastStringToBoolean2()
testCastStringToBoolean3()
testCastStringToBoolean4()
testCastStringToBoolean5()
testCastStringToBoolean6()
testTraceKpi()
AlignmentsExample6_g24_strictTest
example6Fullg24()
example6Patterng24_0_5_n1()
example6Patterng24_0_5_n2()
TestCaseAmtegaBigPatronesSpark
amtega700_071_004()
amtegaStart_706()
amtega700_621_707_004()
amtega700_001()
amtega700_004()
amtega004_700_621_707()
amtega621_707_004_071_700()
AlignmentsExample4SparkTest
example4SparkSometimesexecuteSparkdSubprocess_A_to_E()
example4SparkSometimesexecuteSparkdSubprocess_C_to_E()
example4SparkAlwaysexecuteSparkdSubprocess_A_to_B()
example4SparkAlwaysexecuteSparkdSubprocess_A_to_D()

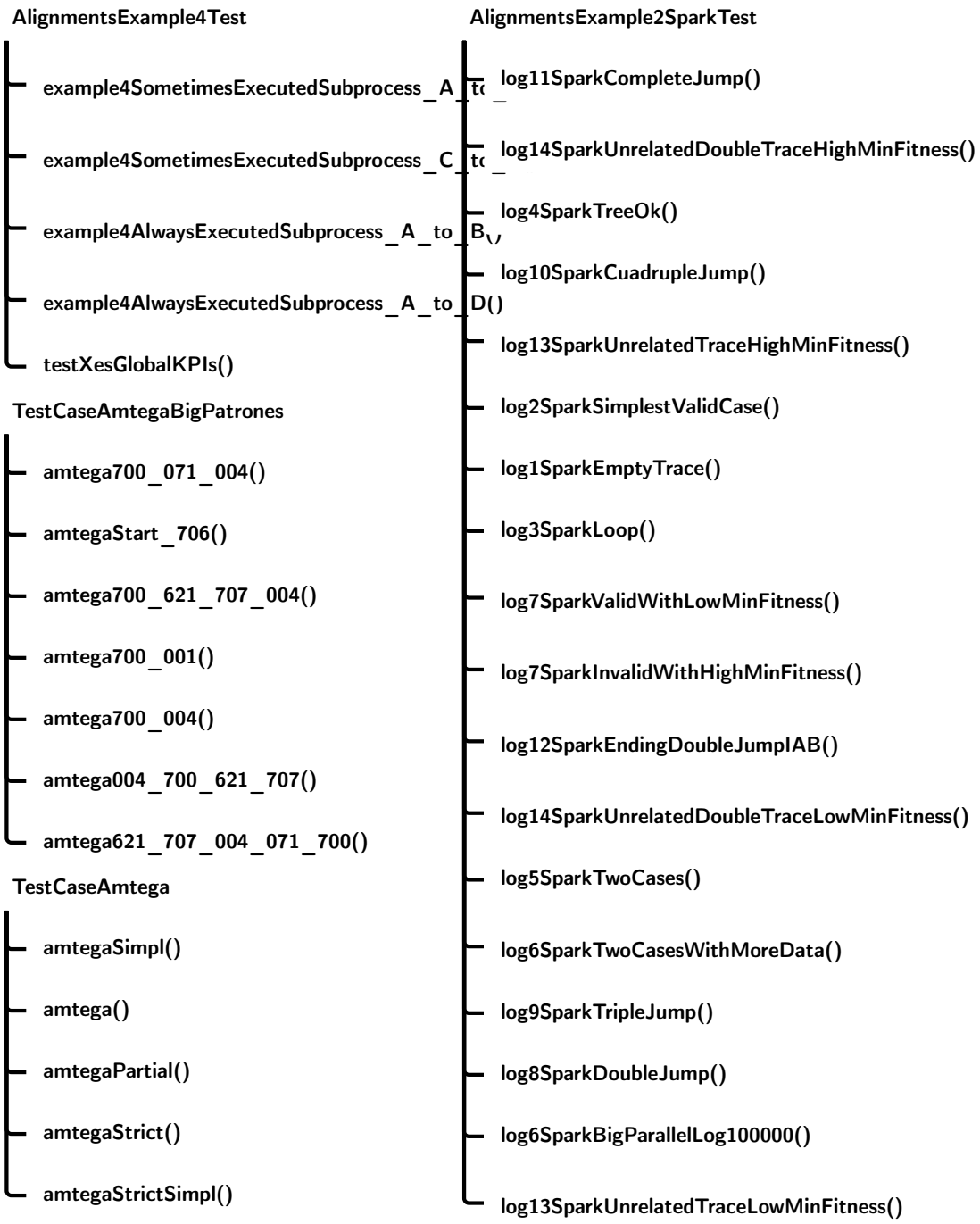

RestrictionTest
applyExample1Log2TimeBasedRestrictionValid
applyExample1Log2TimeBasedRestrictionOnLa
applyExample1Log2TimeBasedRestrictionOnFirst
applyExample1Log2TimeBasedRestrictionInvalid()
applyExample1Log2TimeBasedRestrictionOnFirstTransitionValidFullData()
smallLogMainTest()
applyExample1Log2TimeBasedRestrictionOnFirst
applyExample1Log2TimeBasedRestrictionOnLastTransitionInvalidFullData()
applyExample1Log2TimeBasedRestrictionValidFullData()
applyExample1Log2TimeBasedRestrictionOnLa
applyExample1Log2TimeBasedRestrictionOnLa
applyExample1Log2SimpleRestriction()
applyExample1Log2TimeBasedRestrictionOnFir
applyExample1Log2TimeBasedRestrictionInvalid
AlignmentsExample2Test
log14UnrelatedDoubleTraceHighMinFitness()
log1EmptyTrace()
log12EndingDoubleJumpIAB()
log9TripleJump()
log13UnrelatedTraceHighMinFitness()
log4AlignmentsNeedToJumpInModel()
log13UnrelatedTraceLowMinFitness()
log6BigParallelLog100000Spark()
log7InvalidWithHighMinFitness()
log11CompleteJump()
log14UnrelatedDoubleTraceLowMinFitness()
log6TwoCasesWithMoreData()
log8DoubleJump()
log6BigParallelLog100000()
log2SimplestValidCase()
log7ValidWithLowMinFitness()
log5TwoCases()
log10CuadrupleJump()
log9TracesTripleJump()
log3Loop()
log12TracesEndingDoubleJumpIAB()
log4TreeOk()


StepListenerArcProcessorTest
smallLogMainTest()
AlignmentsExample1SparkTest
log4SparkTwoValidSimpleTraces()
log7SparkAlignmentsOverlappingCases2()
log6SparkAlignmentsOverlappingCases()
log5SparkTraceWithExtraUnnecessaryActivities
log2SparkSimplestValidCase()
log1SparkEmptyTrace()
log5SparkTraceWithExtraUnnecessaryActivi
log3SparkAlignmentsTwoValidCasesNextToEa
log5SparkTraceWithExtraUnnecessaryActivitiesAndDontAllowTraceJumps()
TestCaseNeerlandesesAutomaticPatterns
doTest()
SparkRDDAsListTest
testForeach()
testParallelStream()
testStream()
SubLogReferenceTest
testCreateAndCountTraces()
testCreate()
testCreateSaveAndRestore()
ModelAdapterTest
testFullCircle()
testSaveToFileWithExportedFormat()
DotLanguageTest
invalidNotDigraph()
baseConId()
fullExecutionRestrictionTimeBasedInvalid()
base()
fullExecutionRestrictionTimeBasedValid()
fullExecutionRestrictionTestNotValid()
fullExecutionRestrictionTestValid()
invalid()
LocalLogStreamingToMongoTest
ss()
testStreamToMongoFullCircleXes()
testStreamToMongoFullCircle()


AlignmentsExample1Test
log1EmptyTrace()
log5TraceWithExtraUnnecessaryActivities()
log6AlignmentsOverlappingCases()
log3TreeTwoValidCasesNextToEachOther()
log5TraceWithExtraUnnecessaryActivitiesAndSomeTraceCostLowMinFitness()
log3AlignmentsTwoValidCasesNextToEachO
log7TreeAlignmentsOverlappingCases2()
log4TwoValidSimpleTraces()
log6TreeOverlappingCases()
log2SimplestValidCase()
log7AlignmentsOverlappingCases2()
log3AlignmentsTwoValidCasesNextToEachOther
log4TwoValidSimpleTracesCountArcs()
log5TraceWithExtraUnnecessaryActivitiesAndDontAllowTraceJumps()
log3AlignmentsTwoValidCasesNextToEachOther()
log5TraceWithExtraUnnecessaryActivitiesAndSomeTraceCostHighMinFitness()
SparkLoadFromMongoTest
exampleSparkMongoFull()
exampleSparkMongoSamplePattern()
AlignmentsExample5_g7Test
example5Fullg7()
example5Patterng7_0_6_n3()
example5Patterng7_0_6_n5()
example5Patterng7_0_8_n1()
example5Patterng7_0_8_n2()
example5Patterng7_0_8_n3()
LogMongoSparkAdapterILogTest
testFullCircle()
testSaveToMongoWithSpark()
TestInputToMongo
testInputToMongoSchemaAndSimplify()
testInputToMongo()
testInputToMongoSchema()


MyModelLanguageReaderTest
readCheckForkInSeq()
readCheckSeqLoopCommaSwitchFork()
readCheckSeqSwitchLoopSwitch()
readCheckSeqForkSwitchCommaLoop()
readCheckSeqForkCommaFork()
readCheckSeqForkCommaLoop()
readCheckSeqSimpleEqualsSeqInner()
readCheckSeqSwitchLoopFork()
readCheckSeqSwitch()
readCheckSeqLoopCommaSwitch()
readCheckSwitchFork()
readCheckSeqSwitchCommaSwitch()
readCheckSwitchForkCuadruple()
readCheckLoopSeqSwitch()
readCheckLoopSimple()
readCheckLoopSingle()
readCheckLoopSwitch()
read()
MyModelLanguageReaderTest
readCheckSeqLoopSwitch()
readCheckSwitchAlternativeFork()
readCheckForkFork()
readCheckSeqForkLoopFork()
readCheckSeqForkLoopSwitch()
readCheckSeqForkDoubleSwitchCommaLoop()
readCheckSeqLoop()
readCheckSeqLoopCommaForkSwitch()
readCheckSeqLoopSwitchLoopWithSteps()
readCheckSeqLoopCommaFork()
readCheckSeqLoopCommaLoop()
readCheckSwitchForkInsideSwitchFork()
readCheckSwitchSimple()
readCheckSeqForkCommaSwitch()
readCheckSeqLoopCommaForkDoubleSwitch()
readCheckSwitchForkTriple()
readCheckCompleteExample()
readCheckForkForkBigger()

- MyModelLanguageReaderTest
  - readCheckForkSimple()
  - readCheckForkSwitch()
  - readCheckLoopFork()
  - readCheckLoopLoop()
  - readCheckSwitchForkDoubleFork()
- LogMongoSparkAdapterInVerbisLogTest
  - testFullCircle()
  - testSaveToMongoWithSpark()
- TestCaseBPI2015
  - n1HeuristicCSVSimpl()
  - n2HeuristicLowerMinFitness()
  - n2InductiveSimpl()
  - n5InductiveSimpl()
  - n1HeuristicSimpl()
  - n5Inductive()
  - n2Inductive()
  - n1Inductive()
  - n5HeuristicLowerMinFitness()
  - n2HeuristicSimpl()
  - n1InductiveLowMinFitnessFailsBecauseOfMe[illegible]()
  - n5Heuristic()
  - n5HeuristicSimpl()
  - n2Heuristic()
  - n1InductiveSimpl()
- AlignmentsExample6_g24SparkTest
  - example6Fullg24()
  - example6Patterng24_0_5_n1()
  - example6Patterng24_0_5_n2()
- MLogSparkTest
  - getArrayCaseInstances()
  - getArrayCaseInstances2()
  - testSaveToMongoWithSpark()
- TestCaseAmtegaPatrones
  - amtega700_071_004()
  - amtegaStart_706()
  - amtega700_621_707_004()
  - amtega700_001()
  - amtega700_004()
  - amtega004_700_621_707()
  - amtega621_707_004_071_700()
- LogMongoSparkAdapterCommonLogTest
  - testFullCircle()
  - testSaveToMongoWithSpark()
- LogLocalAdapterTest
  - testFullCircle()
  - testSaveToFileWithExportedFormat()


AlignmentsExample4Test
example4SometimesExecutedSubprocess_A_to_
example4SometimesExecutedSubprocess_C_to_
example4AlwaysExecutedSubprocess_A_to_B()
example4AlwaysExecutedSubprocess_A_to_D()
testXesGlobalKPIs()
TestCaseAmtegaBigPatrones
amtega700_071_004()
amtegaStart_706()
amtega700_621_707_004()
amtega700_001()
amtega700_004()
amtega004_700_621_707()
amtega621_707_004_071_700()
TestCaseAmtega
amtegaSimpl()
amtega()
amtegaPartial()
amtegaStrict()
amtegaStrictSimpl()
AlignmentsExample2SparkTest
log11SparkCompleteJump()
log14SparkUnrelatedDoubleTraceHighMinFitness()
log4SparkTreeOk()
log10SparkCuadrupleJump()
log13SparkUnrelatedTraceHighMinFitness()
log2SparkSimplestValidCase()
log1SparkEmptyTrace()
log3SparkLoop()
log7SparkValidWithLowMinFitness()
log7SparkInvalidWithHighMinFitness()
log12SparkEndingDoubleJumpIAB()
log14SparkUnrelatedDoubleTraceLowMinFitness()
log5SparkTwoCases()
log6SparkTwoCasesWithMoreData()
log9SparkTripleJump()
log8SparkDoubleJump()
log6SparkBigParallelLog100000()
log13SparkUnrelatedTraceLowMinFitness()

Apéndice D

Integración continua

Este fichero YAML muestra la instalación, prueba, generación de informes de resultados y cobertura, despliegue en el repositorio nexus y muchas otras tareas realizadas automáticamente en el CI de forma portable gracias al uso de Docker.

```yaml
---

### COMMON ###

variables:
  # This will suppress any download for dependencies and plugins or upload messages which would clutter the console log.
  # `showDateTime` will show the passed time in milliseconds. You need to specify `--batch-mode` to make this work.
  MAVEN_OPTS: "-Dhttps.protocols=TLSv1.2 -Dmaven.repo.local=$CI_PROJECT_DIR/.m2/repository -Dorg.slf4j.simpleLogger.log.org.apache.maven.cli.transfer.Slf4jMavenTransferListener=WARN -Dorg.slf4j.simpleLogger.showDateTime=true -Djava.awt.headless=true"
  # As of Maven 3.3.0 instead of this you may define these options in `.mvn/maven.config` so the same config is used
  # when running from the command line.
  # `installAtEnd` and `deployAtEnd` are only effective with recent version of the corresponding plugins.
  MAVEN_CLI_OPTS: "--batch-mode --errors --fail-at-end --show-version -DinstallAtEnd=true -DdeployAtEnd=true"

# Cache downloaded dependencies and plugins between builds (specified per task for better performance).
#cache:
#  paths:
#    - .m2/repository
#    - $HOME/.yarn-cache
#    - conformancecheckingfrontend/node_modules
#    - tmp_cache/

services:
  - name: mongo
    command:
    - --replSet
    - replicaTest

stages:
  - algorithm-build
  - algorithm-test
  - algorithm-deploy
```

```
27    - frontend-build
28    - frontend-test
29  #   - frontend-deploy
30    - backend-build
31    - backend-build-artifacts
32    - backend-test
33    - backend-deploy

34  image: alpine # root user for before_script
35  before_script:
36      - mkdir -p "$HOME/.m2"
37      - mkdir -p "$HOME/.m2/repository"
38      - mkdir -p "$CI_PROJECT_DIR/.m2"
39      - mkdir -p "$CI_PROJECT_DIR/.m2/repository"
40      - echo "<settings xmlns=\"http://maven.apache.org/SETTINGS/1.0.0\"
        ↪  xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"
        ↪  xsi:schemaLocation=\"http://maven.apache.org/SETTINGS/1.0.0
        ↪  https://maven.apache.org/xsd/settings-1.0.0.xsd\"><servers><server><id>maven</id><username>${MAVEN_REPO_U
        ↪  > "$HOME/.m2/settings.xml"
41      - echo "<settings xmlns=\"http://maven.apache.org/SETTINGS/1.0.0\"
        ↪  xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"
        ↪  xsi:schemaLocation=\"http://maven.apache.org/SETTINGS/1.0.0
        ↪  https://maven.apache.org/xsd/settings-1.0.0.xsd\"><servers><server><id>maven</id><username>${MAVEN_REPO_U
        ↪  > "$CI_PROJECT_DIR/.m2/settings.xml"
42      - find repo-local-deps/repository/ -name \*.lastUpdated -exec rm -fv {} +
43      - find "$HOME/.m2" -name "_remote.repositories" -type f \! -wholename
        ↪  '*/conformancechecking/*' -delete
44      - find "$CI_PROJECT_DIR/.m2" -name "_remote.repositories" -type f \! -wholename
        ↪  '*/conformancechecking/*' -delete
45      - find "$CI_PROJECT_DIR/repo-local-deps" -name "_remote.repositories" -type f
        ↪  -delete
46      - find "/builds/jacobo.casas/conformancechecking/.m2/repository/" -name
        ↪  "*.part.lock" -type f -delete
47      - cp -r --update repo-local-deps/repository/* "$CI_PROJECT_DIR/.m2/repository"
48      - cp -r --update repo-local-deps/repository/* "$HOME/.m2/repository"
49      # - du "$CI_PROJECT_DIR/.m2/repository"
50      - mkdir -p "$HOME/.gradle"
51      - echo -e
        ↪  "citius_user=${MAVEN_REPO_USER}\ncitius_password=${MAVEN_REPO_PASS}\ncitius_server=http://172.16.244.216"
        ↪  > "$HOME/.gradle/gradle.properties"
52      # Permissions
53      #- chown -R "$(id -u):$(id -u)" repo-local-deps/
54      #- chmod -R 777 "$HOME/.m2"
55      #- chmod -R 777 "$CI_PROJECT_DIR/.m2"
56      #- chmod -R 777 "repo-local-deps/"


57  ### ALGORITHM ###

58  .algorithm-build: &algorithm-build
59    stage: algorithm-build
60    script:
61      - 'cd conformancechecking'
62      - 'mvn $MAVEN_CLI_OPTS install -DskipTests'
63    when: manual
64    cache:
65      policy: pull
66      paths:
67        - .m2/repository
68    artifacts:
69      paths:
70        - .m2/repository/es/usc/citius/alignments/conformancechecking/*
71      expire_in: 1d
72    dependencies: []

73  .algorithm-test: &algorithm-test
```

```
74    stage: algorithm-test
75    script:
76      - 'echo -e "rs.initiate({_id: \"replicaTest\",version: 1,members: [{ _id: 0, host :
        ↪  \"localhost:27017\" }]});\nsleep(30000);\nshow dbs;\nuse
        ↪  spark;\ndb.mongo_needs_this_database_created.insert({\"name\":\"test\"});\nuse
        ↪  conf_chk_local;\ndb.adminCommand( { setFeatureCompatibilityVersion: \"4.0\" }
        ↪  );\ndb.createCollection(\"logs\"); db.createCollection(\"models\");
        ↪  db.createCollection(\"restrictions\"); db.createCollection(\"executions\");
        ↪  db.createCollection(\"execution_results\"); db.createCollection(\"users\")" >
        ↪  mongo_init_script.js'
77      - apt -y update
78      - apt -y install mongodb-clients
79      - 'mongo mongodb://mongo:27017 <mongo_init_script.js'
80      - 'cd conformancechecking'
81      - 'mvn $MAVEN_CLI_OPTS clean verify site 2>&1 | grep -i error'
82    cache:
83      policy: pull-push
84      paths:
85        - .m2/repository
86    artifacts:
87      paths:
88        - conformancechecking/target/site/ # Includes test coverage report and full
          ↪  information about maven
89      expire_in: 1d
90      reports:
91        junit:
92          - conformancechecking/target/surefire-reports/TEST-*.xml
93    dependencies: []

94  .algorithm-deploy: &algorithm-deploy
95    stage: algorithm-deploy
96    script:
97      - 'cd conformancechecking'
98      - mvn $MAVEN_CLI_OPTS deploy -DskipTests
99    allow_failure: true # Nexus repository may be full, and jar is easy to recreate.
100   cache:
101     policy: pull
102     paths:
103       - .m2/repository
104   dependencies: []

105 algorithm-build-jdk8:
106   <<: *algorithm-build
107   image: maven:3-jdk-8
108 algorithm-build-jdk11:
109   <<: *algorithm-build
110   image: maven:3-jdk-11

111 algorithm-test-jdk8:
112   <<: *algorithm-test
113   image: maven:3-jdk-8
114 # algorithm-test-jdk11: # Spark does not support JDK >= 9, even if it compiles correctly
    ↪  (Unsafe usages)
115 #   <<: *algorithm-test
116 #   image: maven:3-jdk-11

117 algorithm-deploy:
118   <<: *algorithm-deploy
119   image: maven:3-jdk-8


120 ### BACKEND ###

121 .backend-build-server: &backend-build-server
122   stage: backend-build
123   script:
```

```
124       - cd conformancechecking
125       - 'mvn $MAVEN_CLI_OPTS install -DskipTests'
126       - cd ../conformancecheckingbackend
127       - ./gradlew bootServerJar
128       - cd ..
129       - mkdir -p tmp_cache/
130       - split -a 1 --numeric-suffixes=1 --bytes=99M --additional-suffix=.jar
          ↪  conformancecheckingbackend/conformancecheckingserver/build/libs/*.jar
          ↪  tmp_cache/backend-build-server.part
131     when: manual
132     cache:
133       policy: pull-push
134       paths:
135         - .m2/repository
136         - tmp_cache
137   #   artifacts:
138   #     paths:
139   #       - conformancecheckingbackend/build/libs/*.jar
140   #     expire_in: 1d
141     dependencies:
142       - frontend-build

143   .backend-build-cluster: &backend-build-cluster
144     stage: backend-build
145     script:
146       - cd conformancechecking
147       - 'mvn $MAVEN_CLI_OPTS install -DskipTests'
148       - cd ../conformancecheckingbackend
149       - ./gradlew bootClusterJar
150       - cd ..
151       - mkdir -p tmp_cache/
152       - split -a 1 --numeric-suffixes=1 --bytes=99M --additional-suffix=.jar
          ↪  conformancecheckingbackend/conformancecheckingcluster/build/libs/*.jar
          ↪  tmp_cache/backend-build-cluster.part
153     when: manual
154     cache:
155       policy: pull-push
156       paths:
157         - .m2/repository
158         - tmp_cache
159   #   artifacts:
160   #     paths:
161   #       - conformancecheckingbackend/build/libs/*.jar
162   #     expire_in: 1d
163     dependencies:
164       - frontend-build

165   # TODO: Add optimization to obfuscation.
166   .backend-build-obfuscated-server: &backend-build-obfuscated-server
167     stage: backend-build
168     script:
169       - cd conformancechecking
170       - 'mvn $MAVEN_CLI_OPTS install -DskipTests'
171       - cd ../conformancecheckingbackend
172       - ./gradlew buildServerObfuscatedJar || true # Expected first time error, and it
          ↪  works the second time
173       - ./gradlew buildServerObfuscatedJar
174       - cd ..
175       - mkdir -p tmp_cache/
176       - split -a 1 --numeric-suffixes=1 --bytes=99M --additional-suffix=.jar
          ↪  conformancecheckingbackend/conformancecheckingserver/build/libs/*-obf.jar
          ↪  tmp_cache/backend-build-obfuscated-server.part
177     when: manual
178     cache:
179       policy: pull-push
180       paths:
```

```
181         - .m2/repository
182         - tmp_cache
183   #   artifacts:
184   #     paths:
185   #       - conformancecheckingbackend/build/libs/*.jar
186   #     expire_in: 1d
187     dependencies:
188       - frontend-build

189   .backend-build-obfuscated-cluster: &backend-build-obfuscated-cluster
190     stage: backend-build
191     script:
192       - cd conformancechecking
193       - 'mvn $MAVEN_CLI_OPTS install -DskipTests'
194       - cd ../conformancecheckingbackend
195       - ./gradlew buildClusterObfuscatedJar || true # Expected first time error, and it
          ↪  works the second time
196       - ./gradlew buildClusterObfuscatedJar
197       - cd ..
198       - mkdir -p tmp_cache/
199       - split -a 1 --numeric-suffixes=1 --bytes=99M --additional-suffix=.jar
          ↪  conformancecheckingbackend/conformancecheckingcluster/build/libs/*-obf.jar
          ↪  tmp_cache/backend-build-obfuscated-cluster.part
200     when: manual
201     cache:
202       policy: pull-push
203       paths:
204         - .m2/repository
205         - tmp_cache
206   #   artifacts:
207   #     paths:
208   #       - conformancecheckingbackend/build/libs/*.jar
209   #     expire_in: 1d
210     dependencies:
211       - frontend-build

212   .backend-upload: &backend-upload
213     image: alpine
214     stage: backend-build-artifacts
215     script: echo "Here! $(ls tmp_cache), getting ${CI_JOB_NAME}"
216     cache:
217       policy: pull
218       paths:
219         - tmp_cache
220     when: manual # Set to manual as dependencies don't make job wait for required data.
221     artifacts:
222       paths:
223         - "tmp_cache/${CI_JOB_NAME}"
224       expire_in: 1d
225   # dependencies: [] - Will be set by specific job

226   .backend-test: &backend-test
227     stage: backend-test
228     script:
229       - cd conformancechecking
230       - 'mvn $MAVEN_CLI_OPTS install -DskipTests'
231       - cd ../conformancecheckingbackend
232       - jar tf ../.m2/repository/es/usc/citius/alignments/conformancechecking/*/*.jar ||
          ↪  true
233       - 'echo -e "rs.initiate({_id: \"replicaTest\",version: 1,members: [{ _id: 0, host :
          ↪  \"localhost:27017\" }]});\nsleep(30000);\nshow dbs;\nuse
          ↪  spark;\ndb.mongo_needs_this_database_created.insert({\"name\":\"test\"});\nuse
          ↪  conf_chk_local;\ndb.adminCommand( { setFeatureCompatibilityVersion: \"4.0\" }
          ↪  );\ndb.createCollection(\"logs\"); db.createCollection(\"models\");
          ↪  db.createCollection(\"restrictions\"); db.createCollection(\"executions\");
          ↪  db.createCollection(\"execution_results\"); db.createCollection(\"users\")" >
          ↪  mongo_init_script.js'
```

```
234       - "SUDO=''; if [ \"$EUID\" -ne 0 ]; then export SUDO='sudo'; fi; $SUDO apt -y update
          ↪  && $SUDO apt -y install mongodb-clients"
235       - 'mongo mongodb://mongo:27017 <mongo_init_script.js'
236       - 'export SPRING_APPLICATION_JSON="{\"mongoloc\": {\"host\": \"mongo\",
          ↪  \"remoteHost\": \"mongo\"}}"'
237       - env # Print environment variables which may affect Spring
238       - ./gradlew -i clean test jacocoTestReport install
239     cache:
240       policy: pull-push
241       paths:
242         - .m2/repository
243     artifacts:
244       paths: # Includes JaCoCo, Dot & test count report.
245         - "conformancecheckingbackend/*/build/reports"
246         - "conformancecheckingbackend/conformancecheckingserver/build/reports"
247       expire_in: 1d
248   #    reports:
249   #      junit: conformancecheckingbackend/*/build/reports/jacoco/**/TEST-*.xml
250     dependencies:
251       - frontend-build

252   .backend-deploy-server: &backend-deploy-server
253     stage: backend-deploy
254     script:
255       - cd conformancechecking
256       - 'mvn $MAVEN_CLI_OPTS install -DskipTests'
257       - cd ../conformancecheckingbackend
258       - ./gradlew :conformancecheckingserver:uploadArchives
259     allow_failure: true # Nexus repository may be full, and jar is easy to recreate.
260     cache:
261       policy: pull
262       paths:
263         - .m2/repository
264     dependencies:
265       - frontend-build

266   .backend-deploy-cluster: &backend-deploy-cluster
267     stage: backend-deploy
268     script:
269       - cd conformancechecking
270       - 'mvn $MAVEN_CLI_OPTS install -DskipTests'
271       - cd ../conformancecheckingbackend
272       - ./gradlew :conformancecheckingcluster:uploadArchives
273     allow_failure: true # Nexus repository may be full, and jar is easy to recreate.
274     cache:
275       policy: pull
276       paths:
277         - .m2/repository
278     dependencies:
279       - frontend-build

280   # backend-build-jdk8:
281   #   <<: *backend-build
282   #   image: gradle:jdk8
283   backend-build-server-jdk:
284     <<: *backend-build-server
285     image: openkbs/jdk-mvn-py3
286   backend-build-cluster-jdk:
287     <<: *backend-build-cluster
288     image: openkbs/jdk-mvn-py3
289   backend-build-server-jdk11:
290     <<: *backend-build-server
291     image: openkbs/jdk11-mvn-py3
292   backend-build-cluster-jdk11:
293     <<: *backend-build-cluster
294     image: openkbs/jdk11-mvn-py3
```

```
295  backend-build-obfuscated-server-jdk8:
296    <<: *backend-build-obfuscated-server
297    image: openkbs/jdk-mvn-py3
298  backend-build-obfuscated-cluster-jdk8:
299    <<: *backend-build-obfuscated-cluster
300    image: openkbs/jdk-mvn-py3

301  backend-build-server.part1.jar:
302    <<: *backend-upload
303  backend-build-server.part2.jar:
304    <<: *backend-upload
305  backend-build-server.part3.jar:
306    <<: *backend-upload
307  backend-build-server.part4.jar:
308    <<: *backend-upload
309  backend-build-server.part5.jar:
310    <<: *backend-upload

311  backend-build-cluster.part1.jar:
312    <<: *backend-upload
313  backend-build-cluster.part2.jar:
314    <<: *backend-upload
315  backend-build-cluster.part3.jar:
316    <<: *backend-upload
317  backend-build-cluster.part4.jar:
318    <<: *backend-upload
319  backend-build-cluster.part5.jar:
320    <<: *backend-upload

321  backend-build-obfuscated-server.part1.jar:
322    <<: *backend-upload
323  backend-build-obfuscated-server.part2.jar:
324    <<: *backend-upload
325  backend-build-obfuscated-server.part3.jar:
326    <<: *backend-upload
327  backend-build-obfuscated-server.part4.jar:
328    <<: *backend-upload
329  backend-build-obfuscated-server.part5.jar:
330    <<: *backend-upload

331  backend-build-obfuscated-cluster.part1.jar:
332    <<: *backend-upload
333  backend-build-obfuscated-cluster.part2.jar:
334    <<: *backend-upload
335  backend-build-obfuscated-cluster.part3.jar:
336    <<: *backend-upload
337  backend-build-obfuscated-cluster.part4.jar:
338    <<: *backend-upload
339  backend-build-obfuscated-cluster.part5.jar:
340    <<: *backend-upload

341  backend-test-jdk8:
342    <<: *backend-test
343    image: openkbs/jdk-mvn-py3
344  #backend-test-jdk11:
345  #  <<: *backend-test
346  #  image: openkbs/jdk11-mvn-py3

347  backend-deploy-server:
348    <<: *backend-deploy-server
349    image: openkbs/jdk-mvn-py3
350  backend-deploy-cluster:
351    <<: *backend-deploy-cluster
352    image: openkbs/jdk-mvn-py3
```

```
353 ### FRONTEND ###

354 frontend-build:
355   image: node
356   stage: frontend-build
357   script:
358     - curl -o- -L https://yarnpkg.com/install.sh | bash
359     - export PATH="$HOME/.yarn/bin:$PATH"
360     - cd conformancecheckingfrontend
361     - yarn install --check-files
362     - yarn run build
363   cache:
364     policy: pull-push
365     paths:
366       - conformancecheckingfrontend/node_modules
367   allow_failure: true # Not critical.
368   artifacts:
369     paths:
370       - conformancecheckingfrontend/dist/*
371     expire_in: 1d
372   dependencies: []

373 frontend-test:
374   image: node
375   stage: frontend-test
376   script:
377     - curl -o- -L https://yarnpkg.com/install.sh | bash
378     - export PATH="$HOME/.yarn/bin:$PATH"
379     - cd conformancecheckingfrontend
380     - yarn install
381     - yarn run test -u
382   cache:
383     policy: pull
384     paths:
385       - conformancecheckingfrontend/node_modules
386   artifacts:
387     paths:
388       - conformancecheckingfrontend/coverage/*
389       - conformancecheckingfrontend/test-report.html
390       - conformancecheckingfrontend/jest_html_reporters.html
391       - conformancecheckingfrontend/jest-stare/*
392     expire_in: 1d
393   dependencies: []
```

Listing 1: Fichero de descripción de integración continua usado por GitLab.

Apéndice E

Glosario

Big Data Es un campo que trata formas de procesar información trabajando con conjuntos de datos demasiado grandes o complejos para ser manejados por el software tradicional de aplicación de procesamiento de datos.

token replay La ejecución basada en marcas hace coincidir una traza y un modelo de red de Petri, comenzando desde el lugar inicial, para descubrir qué transiciones se ejecutan y en qué lugares tenemos tokens restantes o faltantes para la instancia de proceso dada. De esta forma es útil para realizar comprobación de la conformidad tomando como válidas las trazas en las que se llega al lugar final del modelo usado como consulta.

actividad Paso del proceso. Asociado a un registro del *log*. Puede contener información de tiempo en el que se completó cada proceso (inicio y fin), y también el ciclo de vida asociado (inicio, finalización, pausa...).

caso Ver traza.

CD consiste en la automatización de tareas de despliegue que se ejecutan en cada actualización del código..

CI consiste en la automatización de tareas que se ejecutan en cada actualización del código..

clúster Un conjunto de ordenadores coordinados para realizar una tarea en común.

conformidad El análisis de la conformidad es uno de las actividades de minería de procesos. Calcula el número de trazas de los registros que realmente cumplen el modelo que se presenta.

elemento de configuración Cualquier producto de trabajo, tanto producto final como productos intermedios y tanto productos entregables al cliente como productos internos del proyecto, cuyo cambio pueda resultar crítico para el buen desarrollo del proyecto.

evento Ver actividad.

fitness Indica lo bien que se adapta un *log* al modelo. Concretamente, si las trazas se pueden ejecutar en el modelo.

log Secuencia de eventos que ocurren en un proceso.

línea base Conjunto de elementos de configuración formalmente designados y fijados en un momento específico del ciclo de vida. Los elementos incluidos en la línea base tendrán que cumplir unas condiciones mínimas, es decir, han de estar acabados y formalmente aprobados. La línea base sólo puede ser modificada a través de un procedimiento formal de cambios. La línea base, junto con todos los cambios aprobados sobre la misma, representa la configuración vigente y aprobada.

nodo Cada uno de los ordenadores de un clúster.

patrón Lo consideramos un subgrafo del modelo completo en el caso de redes petri, y representa un subconjunto de posibles trazas.

proceso Representamos mediante distintos modelos como redes de Petri o diagramas de estados. En esta memoria, se le llama modelo si se continene todas las posibles trazas y consulta si es un patrón que busca el usuario sobre este modelo.

registro Ver log.

repositorio git carpeta donde se encuentran los documentos asociados al proyecto correctamente organizados con un nombre que los identifica cuyo control de versiones se maneja por la herramienta git.

restriccion Condiciones que se aplican sobre el proceso para dar más libertad a la hora de limitar los resultados. Se expresan en su propio lenguaje de restricciones.

SUS El cuestionario SUS, desarrollado en 1968, ofrece un test sencillo, fácil de puntuar y con el que se obtiene un único número representando la medida de usabilidad del sistema a probar.

tag utilizaremos este término para definir versiones finales que estarán organizadas en carpetas en el repositorio.

TDD es un proceso de desarrollo de software que se basa en la repetición de un ciclo de desarrollo muy corto: primero, el desarrollador escribe un caso de prueba automatizado (inicialmente fallido) que define una mejora deseada o una nueva función, luego produce la cantidad mínima de código para pasar esa prueba, y finalmente refactoriza el nuevo código a estándares aceptables. En este proyecto, se puede escribir código en estándares aceptables directamente para acelerar el proceso.

traza Es un conjunto de eventos que están relacionados según la interpretación que se le dá al registro. Por ejemplo, podrían considerarse como una traza las acciones que realiza un usuario en una sesión.

Bibliografía

[1] IEEE Task Force. Process Mining Manifesto. Disponible en *https://www.win.tue.nl/ieeetfpm/downloads/Process%20Mining%20Manifesto.pdf* consultado el 10 de febrero de 2019.

[2] The Standish Group Report: CHAOS *http://www.projectsmart.co.uk/docs/-chaos-report.pdf*, consultado el 8 de junio de 2019.

[3] Costes Indirectos en Proyectos de I+D de Universidades - USC *http://www.usc.es/export9/sites/webinstitucional/es/congresos/xiiiencontroredeugi/descargas/COS* consultado el 10 de junio de 2019.

[4] Calculadora costes I+D+I USC *http://imaisd.usc.es/ferramentas/calculadora/calculadoracontratos.asp*, consultado el 8 de junio de 2019.

[5] Balsamiq Mockups. (*https://balsamiq.com/*). consultado el 4 de mayo de 2019.

[6] W.P.M. van der Aalst, *Process Mining: Discovery, Conformance and Enhancement of Business Processes*, 1st edition, Springer, 2011.

[7] S. Schönig, C. Di Ciccio, F. M. Maggi, J. Mendling, *Discovery of Multi-perspective Declarative Process Models*, Springer, 2016.

[8] MongoDB, Inc., *MongoDB: Open Source Document Database*, *https://www.mongodb.com/*, consultado el 17 de febrero de 2019.

[9] Abramov, D., *Redux · A Predictable State Container for JS Apps*, *https://redux.js.org/*, consultado el 17 de febrero de 2019.

[10] Apache, *Apache SparkTM - Unified Analytics Engine for Big Data. https://spark.apache.org*, consultado el 17 de febrero de 2019.

[11] Facebook, *React – A JavaScript library for building user interfaces*, *https://reactjs.org/*, consultado el 10 de febrero de 2019.

[12] S. Hernández, Pedro Álvarez, Javier Fabra, Joaquin Ezpeleta, *Analysis of Users' Behavior in Structured e-Commerce Websites*, IEEE Access, 5, 2017.

[13] Graphviz, *The DOT Language*, *https://www.graphviz.org/doc/info/lang.html*, consultado el 10 de febrero de 2019.

[14] Dagre, *Graph layout for JavaScript*, *https://github.com/dagrejs/dagre*, consultado el 10 de febrero de 2019.

[15] JetBrains, *IntelliJ IDEA: The Java IDE for Professional Developers by JetBrains*, *https://www.jetbrains.com/idea/*, consultado el 10 de febrero de 2019.

[16] JetBrains, *Kotlin Programming Language*. *https://kotlinlang.org/*, consultado el 10 de febrero de 2019.

[17] Project Management Institute, Inc., *La guía de los fundamentos para la dirección de proyectos (Guía del PMBOK)*, Project Management Institute, Inc., Sexta edición, 2017.

[18] Parr, T, *ANTLR*. *https://www.antlr.org/index.html*, consultado el 25 de febrero de 2019.

[19] Pivotal, *Spring*. *http://spring.io/*, consultado el 17 de febrero de 2019.

[20] Pivotal, *Spring Boot*. *http://spring.io/projects/spring-boot*, consultado el 17 de febrero de 2019.

[21] Pivotal, *PlantUML*. *https://plugins.jetbrains.com/plugin/7017-plantuml-integration*, consultado el 10 de junio de 2019.

[22] Durán Toro, Amador, *Requirements Management Tool*, *http://www.lsi.us.es/descargas/descarga_programas.php?id=3&lang=en*, Universidad de Sevilla, 2004, consultado el 8 de junio de 2019.

[23] yWorks, *yFiles*. *https://www.yworks.com/products/yfiles*, consultado el 10 de junio de 2019.

[24] Oracle, *Java Streams API*. *https://docs.oracle.com/javase/8/docs/api/java/util/stream/package-summary.html*, consultado el 14 de junio de 2019.

[25] SMARTBEAR, *OpenAPI Specification - Swagger*. *https://swagger.io/docs/specification/about/*, consultado el 14 de junio de 2019.

[26] Instituto Nacional de Tecnologías de la Comunicación, *Guía avanzada de gestión de configuración*, INTECO, 2008.